MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

RUN FREEBSD AS NAT INSTANCE IN CLOUD

INSIDE

VOL.6 NO.11
ISSUE 11/2012(40)
1898-9144

FreeNAS® 8 POWERED

TrueNAS™

UNIFIED. SCALABLE. FLEXIBLE.

Across all industries the demands of data infrastructure have soared to new heights.

As capacity requirements continue to rise at an ever-increasing rate, performance must not be compromised. The hybrid architecture and advanced software capabilities of the TrueNAS appliance enable users to be more agile, effectively manage the explosion of unstructured data and deploy a centralized information storage infrastructure. Whether it's backing virtual machines, business applications, or web services, there's a TrueNAS appliance suited to the task.

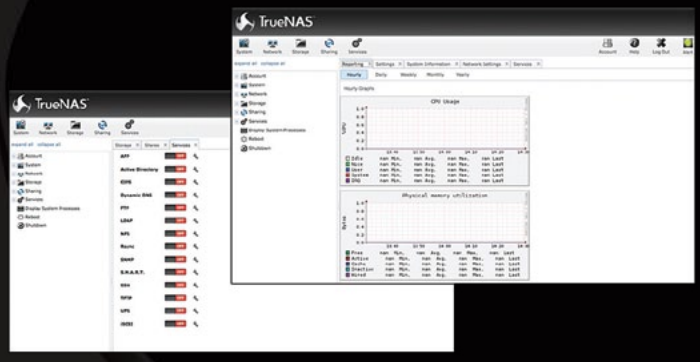
TrueNAS™ Storage Appliances: Harness The Cloud

iXsystems' TrueNAS Appliances offer scalable high-throughput, low latency storage

All TrueNAS Storage Appliances feature the Intel® Xeon® Processors 5600 series, powering the fastest data transfer speeds and lowest latency possible. TrueNAS appliances come in three lines: Performance, Archiver, & High Availability. High-performance, high-capacity ioMemory modules from Fusion-io are available in the TrueNAS Enterprise, Ultimate, and Archiver Pro models.

Key Features:

- **One or Two Six-Core Intel® Xeon® Processors 5600 series**
- **Share Data over CIFS, NFS and iSCSI**
- **Hybrid storage pool increases performance and decreases energy footprint**
- **128-bit ZFS file system with up to triple parity software RAID**



*Optional component

	TrueNAS Pro	TrueNAS Enterprise	TrueNAS Ultimate	TrueNAS Fileshare	TrueNAS Archiver Pro	TrueNAS Pro-HA	TrueNAS Enterprise-HA	TrueNAS Ultimate-HA
	PERFORMANCE			**ARCHIVER**		**HIGH AVAILABILITY**		
Fusion-io Card		X	X		X			
Deduplication					X			
High Availability						X	X	X
Gigabit NICs	Quad	Dual	Dual	Dual	Dual	Quad	Quad	Dual
10 Gigabit NICs		Dual*	Quad*		Dual*			Dual
Max Main Memory	48Gb	96Gb	192Gb	48Gb	192Gb	48Gb	96Gb	192Gb
Max Capacity	220TB	500TB	450TB	580TB	2.2PB	250TB	310TB	1.4PB
Rack Units	2U	2U/4U	4U	2U	4U	3U	3U	Dual 3U

Call iXsystems toll free or visit our website today!

1-855-GREP-4-IX | www.iXsystems.com

Dear Readers,

It's getting colder and rainy outside. We choose the computer more often than a stroll after work. Because of the weather we choose a chat with friends via skype, facebook, or other instead of going out. We are lazy and choose to rely on our Internet connection. No wonder that even the BSD Magazine's articles published this month oscillate around network topic too. As far as the network is concerned, some firewalls and routing is needed as well.

We begin with testing wireless adapters for BSD systems. Next, Kris will tell you about automating deployment of FreeBSD and PC-BSD systems – so we could afford to be lazy during the Winter time.

In Get Started section you will find a nice intro article titled "Network Concepts, Routing and Firewalls", which is intended not only for beginners, but also for those who would like to remind themselves about the basics and ground their knowledge.

The first "How To" article is about installing and running FreeBSD as a NAT instance in Amazon Virtual Private Cloud. It's a really good and practical tutorial. We are sure you will learn a lot from this one. I can only tell you that the reviewers enjoyed it.

Furthermore, you can read the second part of Luca's PostgreSQL indexing, where he will show you some examples, which he has tested on a PostgreSQL 9.1 cluster.

We close the issue with Rob's third part of his series "FreeBSD Enterprise Search with Apache Solr". This time he will teach you how to search for internal or external websites and effectively manage the results on a large scale.

Please remember to send us your comments and opinions on the articles you've read and the topics you would like to see covered in the magazine. Your opinion matters and helps us to do our job well. Never think otherwise!

Well, this is all for now. I wish you a good read!

Patrycja Przybyłowicz
Editor of BSD Magazine
& BSD Team

MAGAZINE BSD

Editor in Chief:
Ewa Dudzic
ewa.dudzic@software.com.pl

Supportive Editor
Patrycja Przybyłowicz
patrycja.przybylowicz@software.com.pl

Contributing:
Rob Somerville, Kris Moore, Luca Ferrari, Andrew Gauld, Anders Berggren, Andrey aka vand777

Top Betatesters & Proofreaders:
Imad Soltani, Luca Ferrari, Cleiton Alves, Eric Geissinger, Mani Kanth, Olaoluwa Omokanwaiye, Radjis Mahangoe, Zander Hill, Darren Pilgrim

Special Thanks:
Denise Ebery

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl

Senior Consultant/Publisher:
Paweł Marciniak *pawel@software.com.pl*

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

Production Director:
Andrzej Kuca
andrzej.kuca@software.com.pl

Executive Ad Consultant:
Ewa Dudzic
ewa.dudzic@software.com.pl

Advertising Sales:
Patrycja Przybyłowicz
patrycja.przybylowicz@software.com.pl

Publisher :
Software Media Sp. z o.o. SK
ul. Bokserska 1, 02-682 Warszawa
Poland
worldwide publishing
tel: 1 917 338 36 31
www.bsdmag.org

Software Press Sp z o.o. SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: editors@bsdmag.org.

What's New

Developers Corner

Get Started

How To

NETGEAR Universal Wifi Adapter (WNCE2001)

The trend towards increased internet connectivity of media devices (TV's, gaming consoles, DVR's) has brought a work-around for one of few my frustrations with BSD operating systems – the limited support for newer wireless adapters.

Many of these media devices have an ethernet port, but no way to attach a wireless adapter. Several companies have stepped up to this opportunity and have created universal wireless adapters that connect to the ethernet port rather than an expansion port.

Since the device connects to the ethernet port, no driver is needed. Since no driver is needed, these devices should work with BSD operating systems. In this article, I will test Netgear's Universal Wifi Adapter, model WNCE 2001.

The WNCE2001 is a small unit that connects to your computer's ethernet port. For power, you get to choose between using an AC wall adapter and using a USB port – a great option when power outlets are limited. The unit is configured through your internet browser.

The hardware I've chosen for this test is the Zotac ZBox Plus. This computer does not have an internal DVD drive, so I will attach an ASUS external DVD drive, model SDRW-08D2S-U, which is powered through a second USB port.

Since the four rear USB ports are now being used by the keyboard, mouse and DVD drive, I will power the WNCE2001 using the AC wall adapter.

To test the WNCE2001, I will preconfigure the device using a live CD of GhostBSD LXDE 3.0 RC1.

Step 1

I created a bootable CD of GhostBSD LXDE 3.0 RC1. You can find download links for GhostBSD images at *http://www.ghostbsd.org/*.

Step 2

I assembled the hardware. I attached the WNCE2001 to the ethernet port before I turned the computer on. This was to ensure that the ethernet port received an IP address from the WNCE2001 when it turned on the DHCP client during bootup.

Step 3

I inserted the GhostBSD CD into the external DVD drive and turned the computer on. In order to boot up the Zbox using the external DVD drive, I had to hit the F11 key on the keyboard. The Zbox then presented a list of medium sources. I selected the external DVD drive and hit Enter.

Step 4

Wireless configuration: It took several minutes for the ZBox to boot up to GhostBSD. When it completed, I was presented with the LXDE window manager. I used the menu buttons on the bottom panel and looked for a virtual terminal application. Upon finding and activating LXTerminal, I executed the command `ifconfig -a`. The

output can be viewed in Figure 2. As you can see, the network interface `re0` has been assigned an IP address of 192.168.1.100.

As I mentioned before, the WNCE2001 is configured using an internet browser. I found an icon for the browser SeaMonkey on the bottom panel. After click-

Figure 1. *NetGear Universal Wifi Adapter (WNCE2001)*

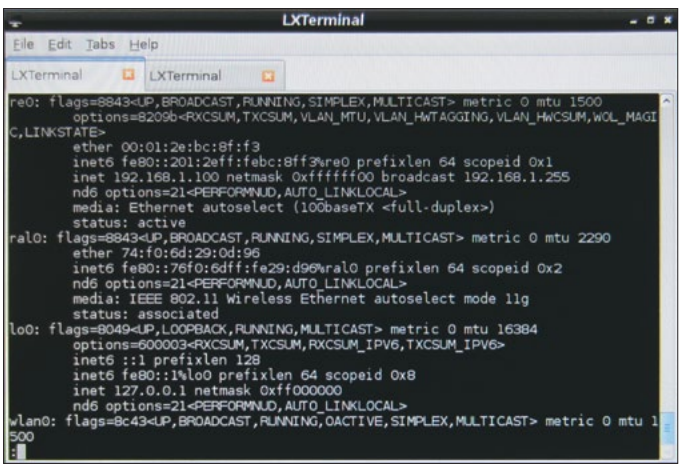


Figure 2. *The ethernet adapter, re0, obtained an IP address via DHCP*

ing on the SeaMonkey icon, SeaMonkey opened to the the NetGear Smart Wizard at *http://www.mywifiext.com/welcome.htm*. On this page, I was prompted to select a language. I accepted the default (English) and clicked 'Continue'.

The configuration tool searched for available networks. On the next page, I was presented a list of networks from which to choose. You can see this list in Figure 3.

After selecting a network and clicking on 'Continue', a webpage appeared showing the security protocol used by the network. I was prompted for a passphrase. I entered the passphrase and clicked 'Continue'. The next page informed me that the WNCE2001 was connecting to the network and that the process would take approximately 2 minutes. Once this was done, a webpage appeared that telling me that the computer was connected to the network successfully. The page also showed the network name, security protocol and the passphrase I had submitted. I clicked on the button, 'Finish' and closed the browser. To test for internet access, I started SeaMonkey again and entered *http://www.ghostbsd.org/* in the url bar. The home page for GhostBSD appeared.

I selected the LXTerminal window and executed the command `ifconfig -a`. The IP address for re0 had been changed to 192.168.1.78.

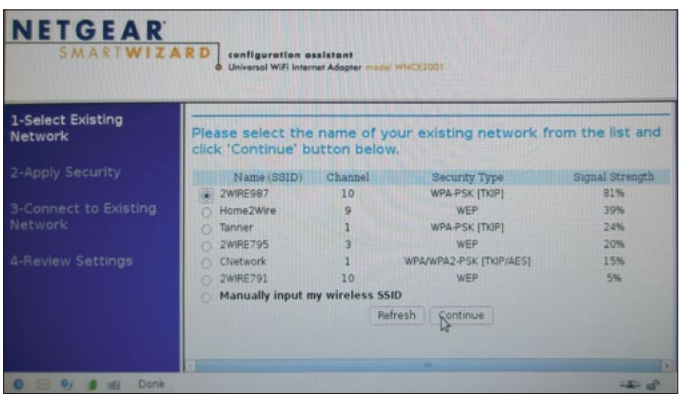


Figure 3. *The configuration wizard displays available networks*

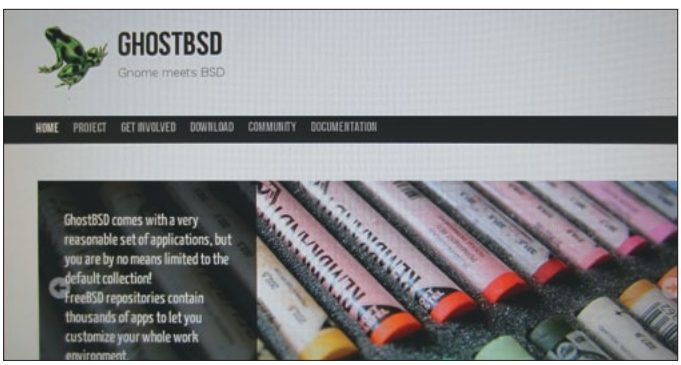


Figure 4. *The computer now has wireless internet access*

Step 5

Curious as to whether I would have to reconfigure the WNCE2001 with every use, even when using the same network, I decided to reboot the computer. Just for fun, I used a Live DVD of PCBSD 9.0. Booting PCBSD 9.0 on the ZBox was uneventful except that the live image was copied into memory, which took several minutes. When it was done, I was given a choice of KDE, Gnome, LXDE or XFCE. I selected LXDE. When LXDE had loaded, I opened the Epiphany web browser. The default page, *http://www.google.com*, appeared. The WNCE2001 did not have to be reconfigured after rebooting the computer.

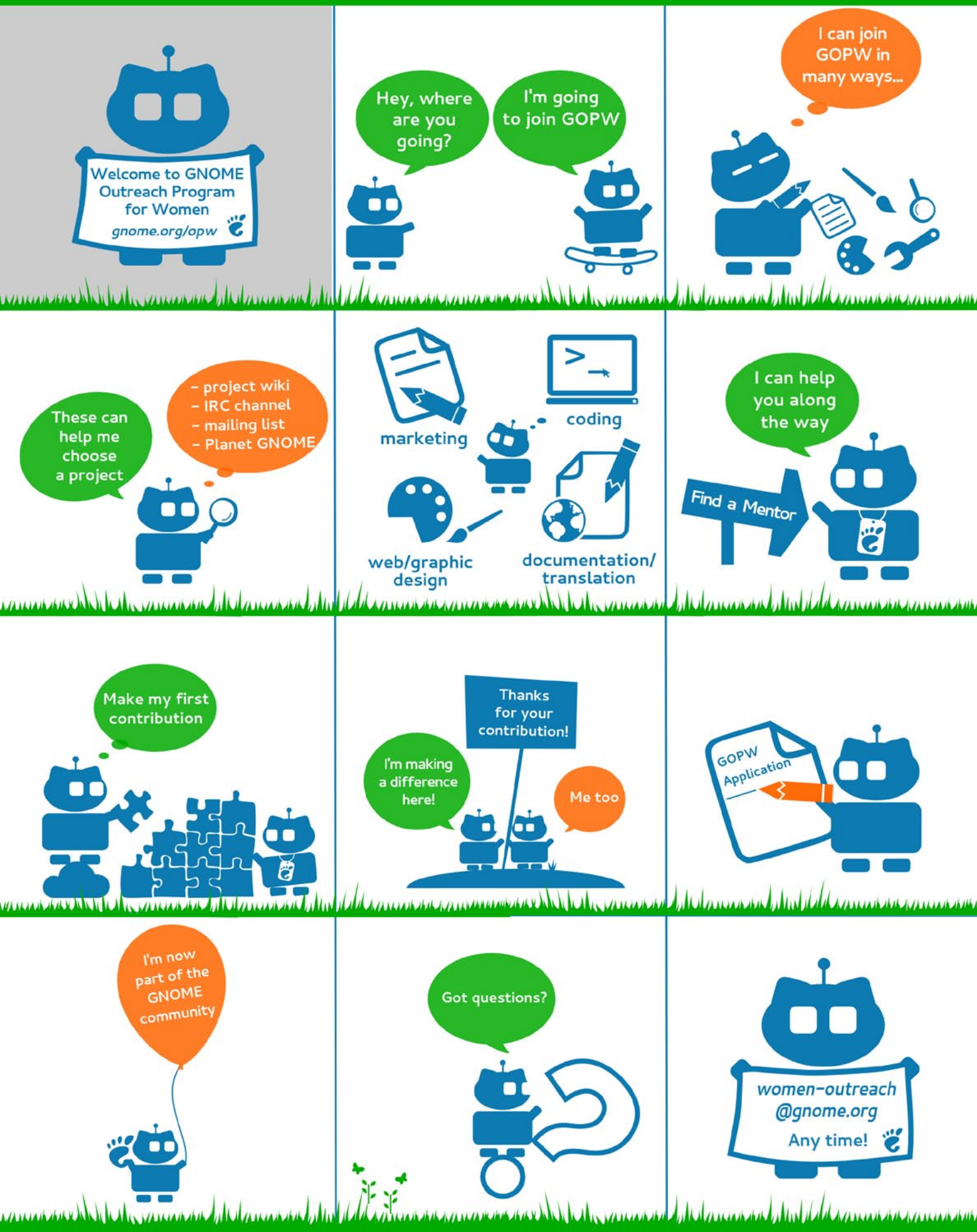
If you ever need to reconfigure the adapter, input *http://www.mywifiext.com/welcome.htm* into the url bar. This was the address of the page that appeared in Step 4 when we configured the adapter for the first time. When I entered it, the first webpage of the NetGear Smart Wizard appeared, as in Step 4 above.

Conclusion

What have we learned? We have established that the use of universal wifi adapters and an ethernet port can by pass driver compatibility issues for wireless network access. We have also established that the ASUS SDRW-08D2S-U and Zotac ZBox Plus (excluding the internal wifi adapter) are compatible with GhostBSD 3.0 RC1 and PCBSD 9.0.

ANDREW GOULD

Andrew Gould is a Senior Revenue Analyst in the healthcare industry. He has been using FreeBSD, PostgreSQL and Python for over ten years.


Welcome to GNOME Outreach Program for Women
gnome.org/opw
Hey, where are you going?
I'm going to join GOPW
I can join GOPW in many ways...
These can help me choose a project
- project wiki
- IRC channel
- mailing list
- Planet GNOME
marketing
coding
web/graphic design
documentation/ translation
I can help you along the way
Find a Mentor
Make my first contribution
Thanks for your contribution!
I'm making a difference here!
Me too
GOPW Application
I'm now part of the GNOME community
Got questions?
women-outreach @gnome.org
Any time!

Automating

the Deployment of FreeBSD and PC-BSD® Systems

In PC-BSD 9.x every installation is fully-scripted, due to the the pc-sysinstall backend. This backend can also be used to quickly automate the deployment of FreeBSD servers and PC-BSD desktops using a PXE boot environment.

In PC-BSD & TrueOS™ 9.1 and higher, this functionality is easy to setup and deploy using the `pc-thinclient` utility. PXE booting allows you to boot systems via the LAN interface, as opposed to using traditional media, such as DVD or USB. In order for clients to boot via PXE they will need a PXE capable network adapter.

The Initial PXE Setup

To get started, you will need to have a system with two network interfaces running PC-BSD or TrueOS 9.1 and a complete ports tree in `/usr/ports`. If you do not have the ports tree installed, you can download it by running the command *portsnap fetch extract update* as root. With these pieces in place, open a root prompt and run the `pc-thinclient` command. The first screen you see will look something like this: Figure 1.

```
root@pxehost:/root # pc-thinclient
/usr/local/bin/pc-thinclient will install the components to convert this system
into a thin-client server.
Continue? (Y/N)
```

Figure 1. *Starting the pc-thinclient wizard*

```
Do you wish to make this a remote X desktop server or install server?
(r/i)
```

Figure 2. *Selecting the type of server to setup*

```
Setting up system for PXE booting...
What NIC do you wish DHCPD to listen on? (I.E. re0)
nic)
```

Figure 3. *Selecting the NIC for DHCPD*

Enter “y” to continue, and the following screen will be shown: Figure 2.

In this case you are going to be setting up a PXE installation server, so enter “i” to continue. (The “r” option can be used to make your system a X thin-client server. More information about this can be found on the wiki page at the end of the article). After selecting your type of PXE system, the thin-client wizard will then begin to build the

```
To perform system installations, place your custom pc-sysinstall scripts in:
/usr/home/thinclient/installscripts

An example script is provided in the above directory

For unattended installations, save your pc-sysinstall script as:
/usr/home/thinclient/installscripts/unattended.cfg

Your system is now setup to do PXE booting!
```

Figure 4. *The finish screen for pc-thinclient*

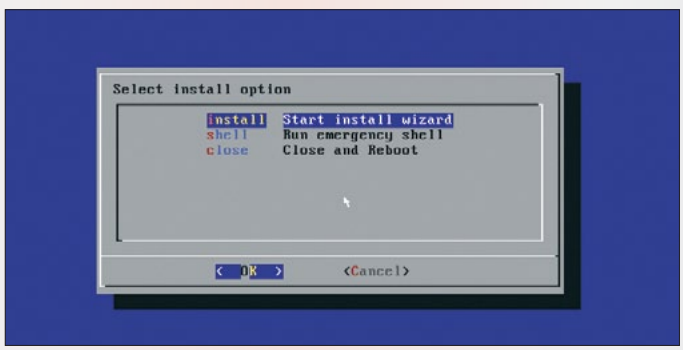


Figure 5. *Booting the client from PXE to the ThinClient Menu*

net/isc-dhcp42-server port. You will most likely only need the default port options, and can hit enter at any confirmation screens that appear. Once the port has finished installing, the thin-client setup will prompt you again for the PXE network interface to use: Figure 3.

Listing 1. *Sample configuration of pc-sysinstall for a PXE client*

```
#Sample configuration file for an installation using
                    pc-sysinstall

installMode=fresh
installInteractive=no
hostname=examplesystem

# Set the disk parameters
disk0=ada0
partition=all
bootManager=none
commitDiskPart

# Setup the disk label
# All sizes are expressed in MB
# Avail FS Types, UFS, UFS+S, UFS+J, ZFS, SWAP
# Size 0 means use the rest of the slice size
disk0-part=UFS+SUJ 1000 /
disk0-part=SWAP 2000 none
disk0-part=UFS+SUJ 0 /usr
commitDiskLabel

# Set if we are installing via optical, USB, or FTP
installType=FreeBSD
installMedium=local
localPath=/installarchive
packageType=tar
installFile=fbsd-release.txz

# Set the root pass
rootPass=root

# Setup our users
# Setup our users
userName=kris
userComment=Kris Moore
userPass=kris
userShell=/bin/csh
userHome=/home/kris
userGroups=wheel,operator
commitUser
```

Enter the interface name of the network card you wish to use as the PXE interface. This interface will be running the DHCP server, and should not be connected to an existing network with another DHCP server running. The wizard will then finish up the configuration for PXE booting, and display a message similar to this: Figure 4.

Your initial PXE setup is now complete! You may now try to PXE boot a client connected on the network interface you specified. If the client boots successfully, you should be presented with an installation screen like Figure 5.

By selecting the "install" option you will be presented with the example configuration option, which can be used to install a basic FreeBSD system. (Below we will look at adding your own) By selecting it, you will be asked to confirm one last time that you wish to perform the installation using this configuration, and then the installation will proceed. In addition to performing installs, you can also access an emergency shell prompt. This can be very useful if you have a system which can no longer boot, and you wish to access the disk or attempt repairs of some kind.

Customizing the Installation Scripts

With your initial PXE configuration finished, you will now most likely want to create your own installation scripts. The thin-client wizard creates an example installation script in the /usr/home/thinclient/installscripts/ directory, which is where you will want to place your custom scripts as well. Any scripts placed in this directory will be selectable as an installation option on the PXE client.

With the location of the install scripts in hand, lets now take a look at the provided pc-sysinstall.example file in that directory (Listing 1).

The included comments in the example make most of the functionality fairly self-explanatory. However there are a few sections we will take a closer look at customizing. The first place to note is the installMedium=local and localPath=/installarchive options. The pc-sysinstall backend supports a number of methods of fetching the archive files for installation. In this case we are using the local file fbsd-release.txz, stored in /usr/home/thinclient/installarchive. This directory will appear to PXE clients

PRO Tip!

By using the *Warden* utility in PC-BSD / TrueOS, it is possible to setup your install environment inside a jail on the host system. Then when you are satisfied with the configuration, you can stop the jail and create the install archive on the host system with the tar command as shown below:

```
# tar cvJf /usr/home/thinclient/installarchive/
myarchive.txz -C /usr/jails/<jailip>
```

as `/installarchive`, allowing you on the host OS to supply your own tar archives of any FreeBSD / PC-BSD release or configuration you wish.

After selecting your installation archive options you will most likely want to customize the disk layout of the new system. In the example provided we are performing a very simple full-disk installation to the disk `ada0`, using UFS with Soft Updates + Journaling. The resulting installation will contain a 1000MB root partition, a 2000MB swap, and use the rest of the disk space for `/usr` as seen Listing 2.

In addition to UFS support, the pc-sysinstall backend can also handle more advanced ZFS disk layouts in a similar manner: Listing 3.

PRO Tip!

Since every single install with pc-sysinstall is fully scripted, a good way to experiment with alternative ZFS layouts is using a virtualization tool such as VirtualBox and your PC-BSD installation DVD. By running the install GUI, you can customize your ZFS disk layout in an easy-to-use manner inside a virtual machine. Then when the installation is finished, a copy of the saved pc-sysinstall configuration file will be saved onto the installed system at `/root/pc-sysinstall.cfg`.

Listing 2. *Adjusting the UFS disk layout for this thin-client*

```
#Set the disk parameters
disk0=ada0
partition=all
bootManager=none
commitDiskPart

# Setup the disk label

# Avail FS Types, UFS, UFS+S, UFS+J, ZFS, SWAP
# Size 0 means use the rest of the slice size
disk0-part=UFS+SUJ 1000 /
disk0-part=SWAP 2000 none
disk0-part=UFS+SUJ 0 /usr
commitDiskLabel
```

Listing 3. *Switching to ZFS for the default client FS*

```
#Setup the disk label
# All sizes are expressed in MB
# Avail FS Types, UFS, UFS+S, UFS+J, ZFS, SWAP
# Size 0 means use the rest of the slice size
disk0-part=ZFS 0 /,/usr,/var,/root (mirror: ada1)
commitDiskLabel
```

References

Wiki Articles:

- *http://wiki.pcbsd.org/index.php/Thin_Client*
- *http://wiki.pcbsd.org/index.php/Creating_an_Automated_Installation_with_pc-sysinstall*

Mailing Lists & Forums

- *http://lists.pcbsd.org*
- *http://forums.pcbsd.org/*

In the example above we have changed our disk layout from UFS to a single ZFS pool using the entire disk, and added the disk drive ada1 to the resulting zpool as a mirror device. While this example is fairly simplistic much more complex layouts can be created using ZFS, including raidz, dataset options and more.

One last feature to look at with the thinclient utility is the ability to perform completely unattended installations. If you plan on only doing fully-automated installations via this PXE interface, you may do so simply by creating a configuration script named `unattended.cfg` and placing it in the `/usr/home/thinclient/installscripts/` directory alongside the example. When a client first boots it will check for this `unattended.cfg` file, and if found it will automatically use it for installation. Some caution should be taken when using this method, since simply plugging a PXE booting system into the wrong LAN cable could cause it to be re-installed.

Conclusion

We have briefly looked at some of the ways PC-BSD and TrueOS 9.1 make it easy to setup PXE booting and perform rapid deployment of FreeBSD based systems. We have also just begun to scratch the surface of the type of installations that can be performed with the pc-sysinstall backend. If you want to do further research into more complex installations please take a look at the included wiki references on pcbsd.org. We also will be happy to continue the discussion with you on the PC-BSD mailing lists or forums.

KRIS MOORE

Kris Moore is the founder and lead developer of PC-BSD. He lives with his wife and four children in East Tennessee (USA), and enjoys building custom PC's and gaming in his (limited) spare time. kris@pcbsd.org.

Great Specials

On FreeBSD & PC-BSD Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

$39.95
FreeBSD 9.0 Jewel Case CD Set
or FreeBSD 9.0 DVD

$29.95
PC-BSD 9.0 DVD

$49.95 Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.0 DVD

$99.95
The FreeBSD CD **or** DVD Bundle

Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.0 CD or DVD set
FreeBSD Toolkit DVD

FreeBSD 9.0 Jewel Case CD/DVD $39.95

CD Set Contains:

- **Disc 1:** Installation Boot LiveCD (i386)
- **Disc 2:** Essential Packages Xorg, GNOME2 (i386)
- **Disc 3:** Installation Boot LiveCD (amd64)
- **Disc 4:** Essential Packages Xorg, GNOME2 (amd64)

FreeBSD 8.2 CD $39.95
FreeBSD 8.2 DVD $39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.0 ..... $29.95
FreeBSD Subscription, start with DVD 9.0 ..... $29.95
FreeBSD Subscription, start with CD 8.2 ..... $29.95
FreeBSD Subscription, start with DVD 8.2 ..... $29.95

PC-BSD 9.0 DVD (Isotope Edition)

PC-BSD 9.0 DVD $29.95
PC-BSD Subscription $19.95

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) $39.95

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.0 ..... $79.95

PC-BSD 9.0 Users Handbook $24.95

BSD Magazine $11.99

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD & PCBSD Caps $20.00

BSD Daemon Horns $2.00

Bundle Specials!
Save $$$

BSD Magazine
Available Monthly

FreeBSD Mall

For even **MORE** items
visit our website today!

www.FreeBSDMall.com

Network Concepts, Routing and Firewalls

This article is aimed at anyone who wants to learn more about networking, routers and firewalls. We will discuss this topic in terms of a BSD/PF firewall/router.

What you will learn…

- Important networking concepts
- Routing basics
- The PF firewall

What you should know…

- BSD basics
- Network basics

I work as a product designer at a company called Halon, building FreeBSD and OpenBSD based network appliances, and the Internet is one of the things that fascinates me the most. In my experience, a good understanding of the basic principles of IP and networking is actually the most important asset when it comes to master the Internet. Some readers will surely find some of the topics in this article trivial, but I still believe that the full comprehension of those trivial facts is key to making people more confident in the area of networking. Throughout this article, we will learn about IP, networking and BSD-based firewalls and routers. Kind of a BSD routing crash course.

OSI Layers

Let's start by visualizing the oh-so-important network layers. Google the "OSI model" to get the full picture. Because I will discuss IP primarily, I'll only talk about layer 2 and layer 3. When I say "layer 2" you should be thinking about Ethernet, hardware (MAC) addresses and switches. You should also think "broadcast", because it's one of the fundamental features of Ethernet. Whenever you connect computers to the same switch, they can broadcast freely to each other.

When I say "layer 3", you should be thinking IP addresses, routers, firewalls and the Internet. Whenever you have a network-related problem that you find complex or diffuse, try to tear the problem apart and put the different pieces in categories such as the OSI model's layers. Alright, this started out a bit vague, but please read on.

IP Addresses

I'll head directly to the topic of IP addresses, because it connects the topics of layer 2 and layer 3 nicely.

I'll start by describing CIDR notation, which I will use throughout this article. Have you seen a net mask such as 255.255.255.0? You probably have. Anyway, it basically says that the subnet it defines has 256 addresses. Therefore, it's a quite common net mask for home or small business networks, with fewer than 250-something devices. What it really states is that: "on this layer 2 network, everyone with the same IP address except for the last digit may speak with each other directly". In other words, if I'm 192.168.0.2 and you are 192.168.0.3 we can speak directly using layer 2 (a switch; never even considering going via a router) because the first three digits of our IP address are the same. However, 192.168.0.2 cannot speak directly to 192.168.1.3 on a 255.255.255.0 subnet, because their third digit differs. So, what's CIDR then? Well, it's actually just the notation of writing "192.168.0.1/24" instead of "192.168.0.1 with net mask 255.255.255.0". It means the same thing, but is a lot shorter. In the same fashion, 255.255.0.0 (a subnet with 65536 addresses in

it) has prefix length 16. Try Googling "CIDR notation" to read more, or find a translation table between net masks and prefix lengths. Let's proceed with a few bullet points that we can discuss:

- IP addresses are assigned to network interfaces. A network interface can be physical (such as an Ethernet adapter) or logical (such as a VLAN or tunnel).
- The IP address and the net mask (or prefix length, as it's also called) defines the "directly reachable" (layer 2) network in the routing table.

For example, let's say I have a BSD computer/router with two network interfaces, and assign 192.168.0.1/24 to one of them (the device em1, that I call "LAN"). This can be done at runtime by running

```
# ifconfig em1 192.168.0.1/24
```

I also connect this LAN interface with a switch, which has a few computers attached to it. These computers has IP address such as 192.168.0.2, .3, et.c. If I "ping" one of those computers from my "router", it will find the appropriate interface to send out this ping on by first looking at the "layer 2" (directly reachable) part of the routing table. This is evident by running:

```
# netstat -rn -f inet
Destination Gateway Flags Refs Use Mtu Prio Iface
192.168.0/24 link#2 UC 5 0 - 4 em1
...
```

I've actually told the router that the addresses 192.168.0.0 to 192.168.0.255 (defined by my address 192.168.0.1 and the prefix length 24) are directly reachable using layer 2 on the LAN interface, and the router will not even try layer 3 routing to find the destination. Instead, the router makes an ARP request (trying to find the IP address' MAC address) on the LAN interface, and if it gets a response, sends the ping to that MAC address out on the LAN interface. This surely sounds trivial to many, but is actually very important. We can make a few important assumptions from this:

- You should *almost never* assign two address within the same subnet to two (or more) different interfaces, because it confuses the computer. Let's say I've assigned 192.168.0.1/24 to one interface (LAN) and 192.168.0.10/24 to another interface (WAN). They are, as defined by the /24 prefix length, in the same layer 3 subnet, and thus in the same "layer 2 domain". Consequentially, the computer/router doesn't know on which interface to send packets to 192.168.0.X, because it has two choices. The extreme case would be to assign the same IP address to two different interfaces, which of course, is equally forbidden. There are exceptions, but this a good rule of thumb.
- You *almost never* need to create routes to directly connected networks. Too many times I've seen people trying to solve problems by randomly adding routes. If you have an address such as 192.168.0.1/24 on LAN, you don't need to create a route to 192.168.0.0/24, the BSD kernel should have added a proper layer 2 directly connected route for you.

Routing

Anything that isn't directly connected in the sense that we discussed in the previous section, IP routes are needed. The by far most common route is the default gateway, which really is 0.0.0.0/0 in network language. The default route is added like:

```
# route add default 1.1.1.1
```

Here are a few important facts. As usual, they don't always apply, but are good rules of thumb.

- The most specific route is chosen. If you have one default route via your gateway, and a route to 10.0.0.0/16 via another one (router 2), a packet to 10.0.1.1 will be sent to router 2.
- All routes has a gateway, and that gateway has to be directly reachable using layer 2. That is, if you only have an address such as 192.168.0.2/24 (making 192.168.0.X directly reachable) you cannot add a route to 10.1.0.0/24 via 172.16.0.1, because your computer doesn't know how to speak with 172.16.0.1 using layer 2. A route to 10.1.0.0/24 via 192.168.0.1 would be valid, however.
- Routing is uni-directional. Just because you can send packets to me, doesn't mean I have a route back to you via which I can send the responses. One example would be a VPN appliance. Say you have a server (192.168.0.2), connected via a router (192.168.0.3) to the internet. On the same network as the server, you places a VPN appliance (192.168.0.3). To this VPN appliance a roaming user connects, and get's the VPN address 10.0.0.2. The VPN user tries to ping 192.168.0.2. The packet travels over the VPN to the VPN appliance, which sends it to 192.168.0.2. It has reaches it's destination. The server howev-

er, responds by sending a packet to 10.0.0.2 via its default route 192.168.0.1 (because neither the server nor the router know about 10.0.0.0/16 being behind the VPN appliance 192.168.0.3) and the packet is lost. The roaming VPN user doesn't get a ping responses. However, adding a route to 10.0.0.0/16 via 192.168.0.3 on the server or the router would have solved the issue.

A router typically forwards packets, and in most BSD operating systems this is enabled by running

```
# sysctl net.inet.ip.forwarding=1
```

for IPv4. Please note that all of the commands I run has to be made permanent by adding configuration to the appropriate files in /etc. This however is very well documented in the respective BSD's manuals, and not the focus of this article.

Dynamic Routing

For a long time, routers have typically been hardware appliances from companies such as Cisco or Juniper. In recent years however, open source routing has matured to the point of being highly suitable even for demanding installations. A modern computer with good network adapters can easily handle several hundreds of thousands packets per second.

The by far most common application for BGP routing is multi-homing; that is, making your services redundant via multiple ISPs. This of course requires redundant connections, an AS number and provider-independent IP addresses, which could be fairly expensive to get hold of. Ask your ISP for more information. Once that is managed however, the configuration is not at all that complicated. In case you're running OpenBGPD, add to `bgpd.conf`

```
AS your-AS
network your-network
neighbor ISPs-router {
   remote-as ISPs-AS
}
neighbor another-ISP {...
deny filters...
```

Although this is not a copy-and-paste example, it hopefully gives you a glimpse of OpenBGPD's elegance and simplicity. The same goes for OpenOSPFD; it's ospfd.conf format is compact and comprehensible, and the program itself works like a charm.

Aliases Addresses

In case you ask your ISP for multiple address, they might give you either a few addresses, or a whole network which you are expected to be the router for via a link network. In the first case, where you're just given a few addresses, you can add those to your WAN interface (called `em0` in this example) as alias addresses with a all-ones (/32) net mask. The reason to use `/32` is to avoid adding layer 2 routes for the addresses. If `1.1.1.2/27` is your primary IP address, and you're also given `1.1.1.3` and `1.1.1.4` by your ISP, these can be added like:

```
# ifconfig em0 1.1.1.2/27
# ifconfig em0 1.1.1.3/32
# ifconfig em0 1.1.1.4/32
```

The PF Firewall

Most BSDs ship with the *packet filter* (PF); a "stateful" layer 3 (e.g. IP) to layer 4 (e.g. TCP) firewall. It's dual-stack by default, meaning that all rules that doesn't explicitly specify an address family works for both IPv4 and IPv6. This, along with other nice properties of its configuration format, makes it (almost) a joy to create rules for. In fact, I like the PF configuration format that much we decided to use it straight off in our own firewall software. There are a few fundamental concepts which are good to know about:

- The firewall's ruleset is evaluated every time a packet goes in or out of an interface
- Interfaces can be interface groups, hardware ports, VLAN interfaces, etc.
- The last matching rule wins, unless a rule is marked by the quick keyword (I recommend making rules quick)

I personally prefer to do filtering on interface groups instead of interface; both for naming them, and grouping them. I also find it extremely handy that the interface with the default route on it, is automatically added to the "egress" group (which can be considered WAN). To add the `em1` interface to a "lan" group, run:

```
# ifconfig em1 group lan
```

Assuming that the WAN interface is in the egress group, we could create a simple firewall ruleset performing NAT by adding to `/etc/pf.conf`:

```
block log
pass on lan
pass out on egress nat-to (egress)
```

and apply the changes by running

```
# pfctl -f /etc/pf.conf
```

That's a pretty compact ruleset, don't you think? Its beautiful syntax also makes it very clear just what it does. One could add a port forwarding to this setup very simply, by adding to `/etc/pf.conf`

```
pass in on egress to port 80 rdr-to 192.168.0.100
```

Where 192.168.0.100 could be a web server on your LAN.

The, in my experience, most common pitfall is to underestimate the importance of the rules' order. For example, if using mostly "quick" rules (in which case the first matching rule "wins") it's usually appropriate to move more specific rules towards the top, and more general rules towards the bottom. Consider the example below

```
block log
pass quick on lan
pass in quick on lan to (wan) port 80 rdr-to 10.0.0.100
                    # doesn't work
```

The function of the third rule in the example above would be to redirect traffic headed for the WAN interface's IP to an internal server on a DMZ (sometimes referred to as "reflection", being able to access internal resources on external addresses from within the network). That won't work however, because that rule will never match. The rule just above it, the second rule, is more general, and will always "win". The solution would be to switch places of the second and third rule.

Now that you've seen some of PF's elegance, you could go on and read OpenBSD's excellent PF FAQ. The manual page for pf.conf is also great once you get the hang of it.

Debugging and Monitoring

I will not cover debugging in depth, but give you some hints and pointers. The manual pages will do the rest.

As for general network and firewall debugging, tracing packets can be a very quick way of locating errors. Let's say you have a "case" which you expected to work, but which doesn't. You might not reach a web server in a DMZ, but other servers in the DMZ are responding. You don't know what the problem is, or if it's even in the firewall. Anyway, start tracing packets. Make sure you have a device as initiator. In this case, we want do debug an issue only visible from the "outside". Therefore, use a smartphone connected to cellular data and start trying to access the web server while having this command running on the firewall:

```
# tcpdump -n -i em0 port 80 and host 1.1.1.2
```

where `em0` is your external interface and 1.1.1.2 is the IP you're trying to access from the smartphone. If you see traffic, move on the firewall logging

```
# tcpdump -n -e -ttt -i pflog0 port 80
```

and see if it's blocked (requires all your block rules to have the "log" keyword). If it's not blocked, move on the internal interface where you expect the packets to show up:

```
# tcpdump -n -i em1 port 80 and host 192.168.0.100
```

where `em1` is your internal interface and 192.168.0.100 is the IP that you might have done a port forwarding to. If nothing shows up, start looking at the firewall ruleset, and make sure that the desired packets are in fact matches by your port forward. Perhaps take a look at the routing table to make sure that packets to 192.168.0.X are in fact expected to be sent out on `em1`. And so on. If you however do see packets going out on `em1`, but you don't see the responses, start debugging the server. Is it connected? Does it provide any service on port 80? Is the server configured with the firewall as its default gateway? Otherwise the packets aren't routed back.

Certain debugging and monitoring tasks require you to extract information from the system, such as from PF or the BGP process. For this, there are handy little programs called pfctl, bgpctl, ospfctl, et.c. Google those, and you'll find tons of information in manual pages and tutorials.

ANDERS BERGGREN

Anders Berggren lives in Sweden, where he works with a few old friends at a network security company called Halon Security. His whole grown-up life he has been fascinated with operating systems, and has been building appliances based on FreeBSD and OpenBSD for many years. He can be reached at http://www.halon.se.

FreeBSD
as a NAT Instance in Amazon Cloud

This article is intended for beginners wanting to install and run FreeBSD as a NAT instance in Amazon Virtual Private Cloud (Amazon VPC).

What you will learn…
- How to install, configure and update a FreeBSD instance in Amazon Cloud,
- How to configure private and public subnets, routing and security groups in Amazon VPC,
- How to set up FreeBSD as a NAT instance between public and private subnets.

What you should know…
- FreeBSD basics,
- Amazon Cloud basics.

Amazon VPC lets you launch instances in a virtual network that closely resembles a traditional network that you might operate in your own data center. You place publicly accessible servers (for example, web servers, DNS server etc.) into a public-facing subnet, and place your backend systems (databases, application servers etc.) in a private subnet with no Internet access. Instances in the private subnet can access the Internet only by routing their traffic through a NAT instance in a public subnet. In the configuration wizard Amazon offers its "native", Linux based, instance as the only choice for NAT instance. However, it is possible to configure VPC manually and use FreeBSD (or any other operating system) for NAT purposes.

What we will do

- Configure Amazon Virtual Private Cloud (VPC).
- Launch a FreeBSD instance in a public subnet.
- Configure route table for the private subnet.
- Configure firewall for the FreeBSD instance.
- Perform basic configuration of the FreeBSD instance.
- Apply security patches and install a custom kernel.
- Setup OpenVPN.
- Launch a Microsoft Windows instance in the private subnet.
- Test Remote Desktop Connection to the Microsoft Windows instance to prove that our NAT instance works.

As the result of the above steps we will end up with:

- the public (10.0.1.0/24) and private (10.0.2.0/24) subnets,
- a FreeBSD NAT instance with two network interfaces:
  - the first network interface belongs to the public subnet and has the IP address 10.0.1.10,
  - the second network interface belongs to the private subnet and has the IP address 10.0.2.10.
- a Microsoft Windows instance in the private subnet (IP address 10.0.2.20) which can be accessed (Remote Desktop Connection) through our FreeBSD NAT instance. All outgoing traffic of this Microsoft Windows instance goes through the FreeBSD NAT instance.

Step 1.
Configure Amazon Virtual Private Cloud

In this step we will create and configure VPC (10.0.0.0/16): public (10.0.1.0/24) and private (10.0.2.0/24) subnets, an Internet gateway and a route table for the public subnet.

Task 1. Create VPC

Log in into AWS Management Console and then choose the "VPC" item from the "Services" menu: Figure 1.

In the "Navigation" pane, click the "Your VPCs" link, and then click the "Create VPC" button: Figure 2.

Type 10.0.0.0/16 in the "CIDR Block" field and then press the "Yes, Create" button.

Task 2. Create public and private subnets

In order to create public and private subnets, click the "Subnets" link on the "Navigation" pane, and then click the "Create Subnet" button: Figure 3. You can select a desired availability zone as well as specify subnet's IP address block ("CIDR Block" field). You should create 10.0.1.0/24 as your public subnet, and 10.0.2.0/24 as your private subnet.

Task 3. Create an Internet Gateway.

Create and attach an Internet Gateway (click on the "Internet Gateways" on the "Navigation" pane, and then click the "Create Internet Gateway" button). After creation, select the Internet Gateway in the list and then click the "Attach to VPC" button, select your VPC (10.0.0.0/16) and then click on the "Yes, Attach" button.

Task 4. Create a route table for public subnet

Create a new route table for your public subnet (10.0.1.0/24):

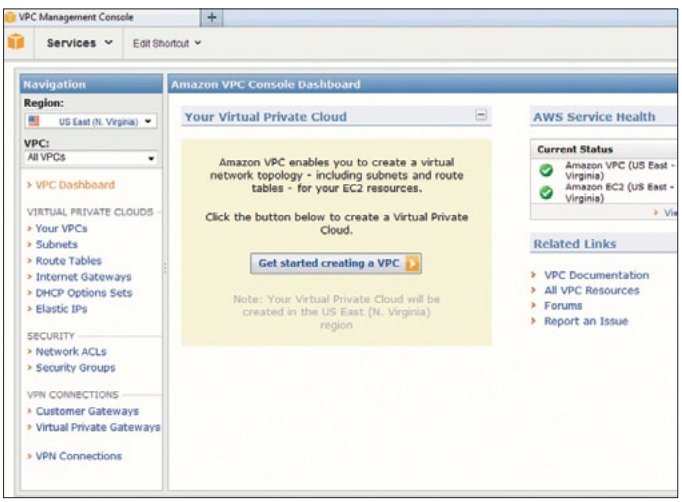


Figure 1. *Amazon VPC Console Dashboard*

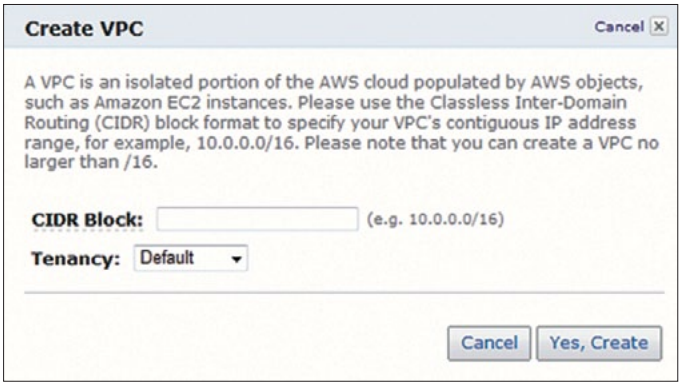


Figure 2. *Create VPC*

- Click on the "Route Tables" on the "Navigation" pane.
- Click on the "Create Route Table" button, select your VPC (10.0.0.0/16) from the drop-down list, and then click "Yes, Create": Figure 4.
- This new route table will appear in the list (it will have "No" in the "Main" column). Select it in the list, and then add a new route (Destination: 0.0.0.0/0, Target: your Internet Gateway) in the last line, and then click "Add": Figure 5.
- Click on the "Associations" tab and then associate this route table with your public subnet (10.0.1.0/24): Figure 6.

Step 2. Launch a FreeBSD Instance in a Public Subnet

I'd like to take this opportunity to thank Colin Percival who creates AMI instances of FreeBSD in Amazon Cloud.

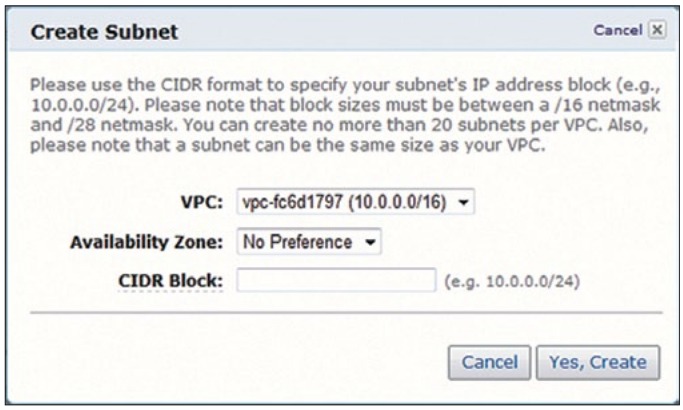


Figure 3. *Create Subnet*

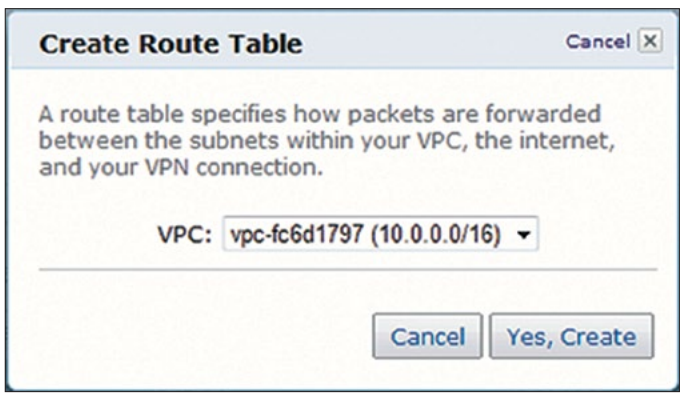


Figure 4. *Create Route Table*

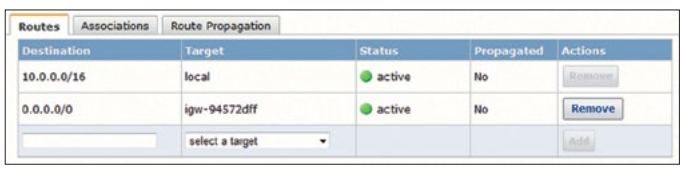


Figure 5. *New Route Table Rules*

You can find available FreeBSD AMIs on his website at *http://www.daemonology.net/freebsd-on-ec2/*. In this step we will:

- Launch a FreeBSD instance and setup its security group,
- Disable source/destination checks (otherwise, NAT will not work),
- Assign an Elastic IP address (EIP) to the NAT instance.

Task 1. Launch a FreeBSD instance.

- Select the "EC2" item from the "Services" menu, and then click on the "Launch Instance" button.
- Choose the "Classic" wizard.
- Click on the "Community AMIs" tab, type the desired AMI from the *http://www.daemonology.net/freebsd-on-ec2/* website in the "Search" field (for example, `ami-1d4c9874` to install FreeBSD 9.0 RELEASE amd64 in US East, North Virginia), and then press ENTER: Figure 7.
- I will install FreeBSD 9.0 RELEASE amd64/HVM as this was the latest release version at the time of this writing.
- On the next wizard step choose the instance type, click on "VPC" and select your public subnet (10.0.1.0/24) from the list. Click "Continue".
- Select "2" in the "Number of Network Interfaces" drop-down list, and assign IP addresses to each interface:
  - 10.0.1.10 for `eth0` (subnet 10.0.1.0/24),
  - 10.0.2.10 for `eth1` (change subnet to 10.0.2.0/24).
- You can assign secondary IP addresses for each interface if needed. Then click "Continue" button twice.
- Name your instance and then click "Continue".
- Create a new key pair or choose the existing one, and then click "Continue".
- Create a new security group (Figure 8):
  - Open port 22 for SSH (TCP),
  - Open port 1194 for OpenVPN (UDP).
- Click "Launch".

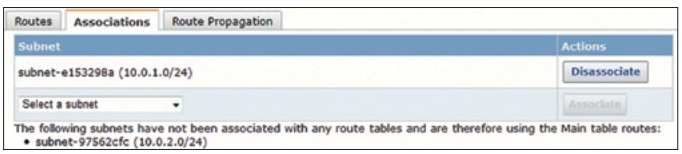


Figure 6. *Route Table Associations*

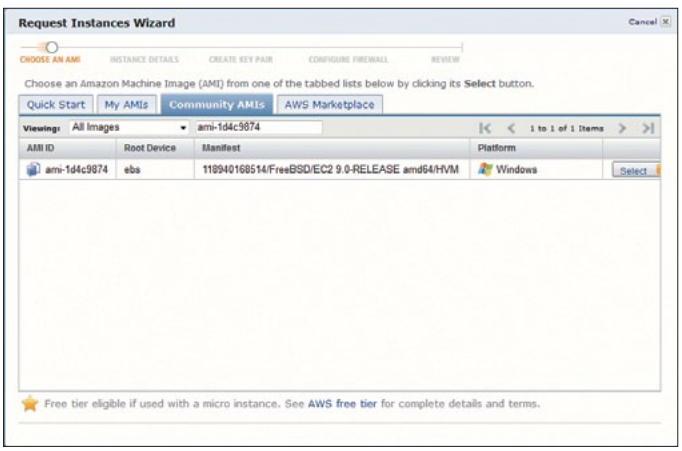


Figure 7. *FreeBSD Community AMIs*

Task 2. Disable source/destination checks on the NAT instance's network interfaces

By default, each instance performs source and destination checking. This means the instance must be the source or destination of any traffic it sends or receives. For a NAT instance to perform network address translation, we must disable source/destination check on our FreeBSD instance:

- Click on the "Network Interfaces" on the "Navigation" pane.
- Right-click on each of the interfaces of your NAT instance and choose the "Change Source/Dest Check" item from the popup menu: Figure 9.
- You should disable source/destination checks on each of the network interfaces of your NAT instance: Figure 10.
- I also usually make a note of the private network interface ID (the one with IP 10.0.2.10). This information will be needed later.

Task 3. Assign an Elastic IP address (EIP)

To access your NAT instance from the Internet, you should assign an *Elastic IP address* (EIP) to your NAT instance:

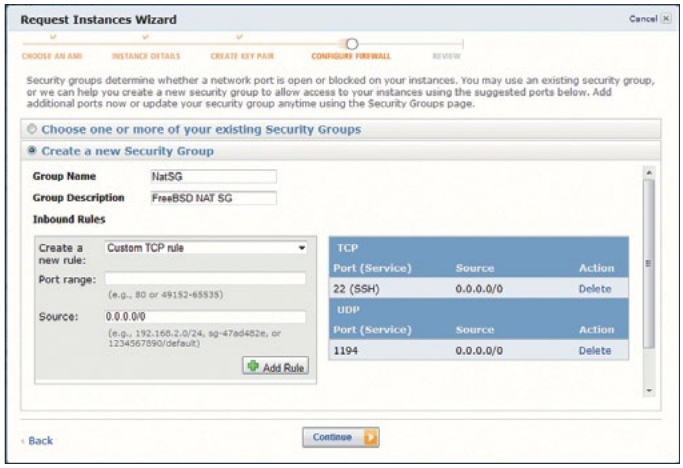


Figure 8. *Create a New Security Group*

- Click on the “Elastic IPs” on the “Navigation” pane.
- Click “Allocate New Address”.
- Select “VPC” from the drop-down list and then click “Yes, Allocate”.
- Select the allocated IP address in the list and then click on the “Associate Address” button.
- Select your NAT instance in the “Instance” drop-down list and select “10.0.1.10*” in the “Private IP address” field. Click “Yes, Associate” button.

Step 3. Configure route table for the private subnet

We will use the main route table to configure routing for the private subnet:

- Select the “VPC” item from the “Services” menu.
- Click on the “Route Tables” on the “Navigation” pane.
- Select the main route table and then add a new route:
  - Destination: 0.0.0.0/0,
  - Target: private interface of your NAT instance (with the IP 10.0.2.10). (We have made notes of the private network interface ID when we were disabling source/destination checks). See Figure 11.
- Click on the “Associations” tab and then associate this route table with your private subnet (10.0.2.0/24): Figure 12.

Now your Amazon VPC configuration is completed and we can focus on configuration of our FreeBSD NAT instance.

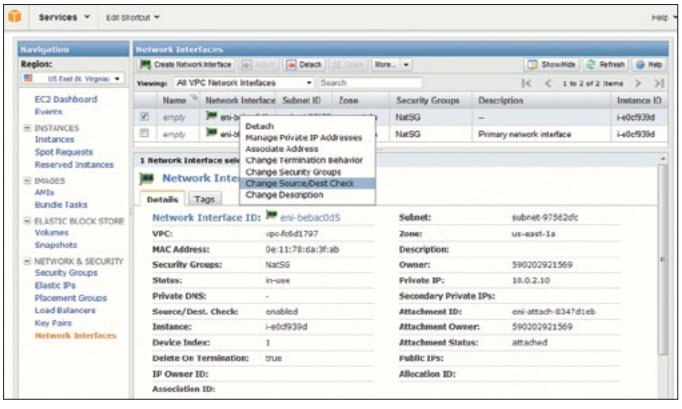


Figure 9. *Change Source/Destination Checks Settings*

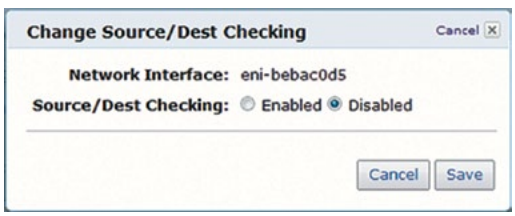


Figure 10. *Disable Source/Destination Checks*

Step 4. Configure Firewall for the FreeBSD Instance

In this step we will enable and configure PF firewall. You need to connect to your FreeBSD instance via SSH as root and then create the `/etc/pf.conf` file: Listing 1.

Then enable PF firewall:

```
# echo 'pf_enable="YES"' >> /etc/rc.conf
# echo 'pf_rules="/etc/pf.conf"' >> /etc/rc.conf
# echo 'pflog_enable="YES"' >> /etc/rc.conf
# /etc/rc.d/pf start
```

Reconnect via SSH.

Step 5. Perform Basic Configuration of the FreeBSD Instance

In this step we will configure our FreeBSD instance:

- Configure network interfaces.
- Add a new user.
- Configure SSH daemon.
- Update ports tree.
- Install openntpd.
- Enable SSHGuard.
- Install tools to manage ports updates and monitor vulnerabilities.

Task 1. Configure network interfaces

Make sure that your server can act as gateway:

```
# echo 'gateway_enable="YES"' >> /etc/rc.conf
```

Configure the 2nd network interface:

```
# echo 'ifconfig_xn1="inet 10.0.2.10/24"' >> /etc/rc.conf
# ifconfig xn1 inet 10.0.2.10/24
```

To improve network performance, it is important to do the following:

```
# echo 'net.inet.tcp.tso=0' >> /etc/sysctl.conf
```

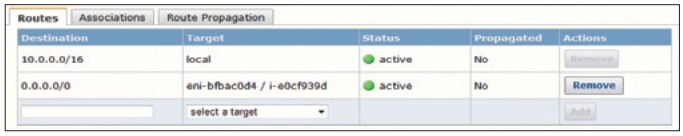


Figure 11. *Main Route Table Rules*

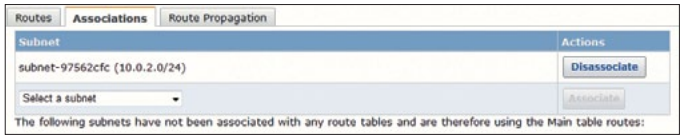


Figure 12. *Main Route Table Associations*

Reboot the server:

```
# reboot
```

Task 2. Add a new user

In order to add a new user, execute the following command:

```
# adduser
```

Please do not forget to "invite" your new user to the *wheel* group. It is more convenient and secure to set-up SSH authorization via a DSA/RSA key than to type the password all the times. Let's do it. Generate DSA keys:

```
# su your_new_user
% ssh-keygen -t dsa
% cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

Listing 1. */etc/pf.conf file contents*

```
ext_if = "xn0"
int_if = "xn1"
loc_if = "lo0"
vpn_if = "tun0"

win_host = "10.0.2.20"

table <private_subnet> { 10.0.2.0/24 }
table <firewall> { self }
table <openvpn_clients> { 10.8.0.0/24 }
table <sshguard> persist
table <jails> { 192.168.50.0/24 }

scrub in all

# NAT
nat on $ext_if inet from <private_subnet> to !<firewall>
                    -> ($ext_if)
nat on $ext_if inet from <jails> to !<firewall> -> ($ext_
                    if)
nat on $int_if from <openvpn_clients> to <private_subnet>
                    -> ($int_if)

block log all
block in quick on $ext_if proto tcp from <sshguard>
                    to ($ext_if) port ssh label "ssh
                    bruteforce"
block in quick on $vpn_if proto tcp from <sshguard>
                    to ($ext_if) port ssh label "ssh
                    bruteforce"

# OpenVPN
pass in on $ext_if inet proto udp from any to ($ext_if)
                    port 1194 keep state

# ssh
pass in on $ext_if proto tcp to ($ext_if) port ssh flags
                    S/SA keep state
pass in on $vpn_if proto tcp from <openvpn_clients> to
                    ($ext_if) port ssh flags S/SA keep
                    state

# RDP
pass in on $vpn_if proto tcp from <openvpn_clients> to
                    $win_host port 3389 flags S/SA keep
                    state

# ICMP to the external IP and OpenVPN gateway IP
icmp_types = "{ echoreq, unreach }"
pass in on $ext_if inet proto icmp from any to ($ext_if)
                    icmp-type $icmp_types keep state
pass in on $vpn_if inet proto icmp from any to ($vpn_if)
                    icmp-type $icmp_types keep state

# pass out this server
pass out proto { tcp, udp, icmp } all keep state

# pass out private subnet
pass in on $int_if proto { tcp, udp, icmp } from
                    <private_subnet> to !<firewall> flags
                    S/SA

# pass out jails
pass in on $loc_if proto { tcp, udp, icmp } from <jails>
                    to !<firewall> flags S/SA

antispoof for $ext_if
antispoof for $int_if
```

Then copy your private key (~/.ssh/id _ dsa) to your Microsoft Windows workstation:

- Print your private key in the SSH console:

```
% cat ~/.ssh/id_dsa.pub
```

- Copy the printed text into a new id _ dsa file on your Microsoft Windows workstation.
- On your Windows workstation, convert id _ dsa key to Putty *.ppk* without a passphrase:
  - Download PuttyGen from *http://the.earth.li/%7Esgtatham/putty/latest/x86/puttygen.exe* and open it.
  - Click the "Load" button, choose "All files (*.*)", and select your id _ dsa key.
  - Click on the "Save Private Key" button.

From now you are able to connect via SSH by using your private key instead of typing your password all the times. But before you do this, let's change root's password:

```
% exit
# passwd
```

Task 3. Configure SSH Daemon

Make sure that SSH daemon listens on 10.0.1.10 only. Add the following line to /etc/ssh/sshd_config:

```
ListenAddress 10.0.1.10
```

Also disable login for root user:

```
-PermitRootLogin yes
+PermitRootLogin no
```

Then restart SSH:

```
# /etc/rc.d/sshd restart
```

Before you close the existing SSH session please check that you can open a new SSH session as the user you added in the previous task.

Task 4. Update ports tree

Update ports tree for the first time:

```
# portsnap fetch extract
```

In future you should update ports tree as below:

```
# portsnap fetch update
```

Task 5. Install Openntpd

OpenNTPD provides the ability to sync the local clock to remote NTP servers and can even act as NTP server itself, redistributing the local clock.

Configure ntpdate to set the clock at boot time:

```
# echo 'ntpdate_enable="YES"' >> /etc/rc.conf
# echo 'ntpdate_hosts="pool.ntp.org"' >> /etc/rc.conf
```

Install net/openntpd:

```
# cd /usr/ports/net/openntpd
# make config-recursive
# make install clean
```

Default configuration will suffice for our purposes but feel free to edit /usr/local/etc/ntpd.conf if needed. Configure openntpd to start at boot time and start it now:

```
# echo 'openntpd_enable="YES"' >> /etc/rc.conf
# /usr/local/etc/rc.d/openntpd start
```

Task 6. Enable SSHGuard

SSHGuard monitors server from its logging activity. When logs convey that someone is trying to brute-force SSH password, SSHGuard reacts by blocking this host.

Let's first install it:

```
# cd /usr/ports/security/sshguard-pf
# make config-recursive
# make install clean
```

Your /etc/syslog.conf has been added a line for SSHGuard; uncomment it and then reload the configuration:

```
# /etc/rc.d/syslogd reload
```

This command will display the set of addresses blocked in the SSHGuard table at any time:

```
# pfctl -Tshow -tsshguard
```

Task 7. Install tools to manage ports updates and monitor vulnerabilities

Install ports-mgmt/portmaster:

```
# cd /usr/ports/ports-mgmt/portmaster
# make config-recursive
```

You should check the box against the PKGNGPATCH option.

```
# make install clean
# echo 'WITH_PKGNG=yes' >> /etc/make.conf
# rehash
# pkg2ng
# cp /usr/local/etc/pkg.conf.sample /usr/local/etc/pkg.conf
```

Edit the `/usr/local/etc/pkg.conf` file:

```
...
PACKAGESITE     : http://pkgbeta.freebsd.org/${ABI}/latest
...
```

Install `ports-mgmt/portaudit`:

```
# cd /usr/ports/ports-mgmt/portaudit
# make config-recursive
# make install clean
# rehash
# portaudit -Fda
```

Step 6. Apply Security Patches and Install a Custom Kernel

FreeBSD is a very secure operating system. However, even in this operating system people sometimes find security holes. (The list of released FreeBSD Security Advisories can be found on the *http://www.freebsd.org/security/advisories.html* webpage).

In this step we will patch all security holes by updating source and then recompiling world and kernel. We cannot use *freebsd-update* utility for these purposes because FreeBSD AMIs use a custom kernel. To recompile the kernel from sources is the only option we have.

Task 1. Synchronise Source

Setup configuration for `csup` utility:

```
# cp /usr/share/examples/cvsup/standard-supfile
/etc/freebsd-supfile
```

Open `/etc/freebsd-supfile` and change the URL of your CVSUP server in the following line:

```
*default host=CHANGE_THIS.FreeBSD.org
```

Note

You can find the list of CVSUP servers on the FreeBSD website.

Save the file and then update the source:

```
# csup /etc/freebsd-supfile
```

Task 2. Apply EC2-specific Patches

Some EC2-specific patches have been applied to FreeBSD in order to improve performance and work around bugs. These are contained in `/root/ec2-bits/*.patch`:

```
# ls /root/ec2-bits/*.patch
/root/ec2-bits/blkfront.patch
/root/ec2-bits/ec2.patch
/root/ec2-bits/tcp_mbuf_chain_limit.patch
/root/ec2-bits/uart.patch
```

Let's apply this patches:

```
# cd /usr/src
# patch < /root/ec2-bits/blkfront.patch
# patch < /root/ec2-bits/ec2.patch
# patch < /root/ec2-bits/tcp_mbuf_chain_limit.patch
# patch < /root/ec2-bits/uart.patch
```

Task 3. Create a Custom Kernel Configuration

For 32-bit systems, you can find the kernel configuration files in the `/usr/src/sys/i386/conf` directory. However, we have installed a 64-bit operation system, so let's use configuration files from the `/usr/src/sys/amd64/conf` directory:

```
# cd /usr/src/sys/amd64/conf
```

I always strip kernel from the drivers I do not use. You should copy the `GENERIC` file into `MY_GENERIC` and then comment those drivers you do not need in your kernel. If you do not know what to comment, then leave everything as it is.

```
# cp GENERIC MY_GENERIC
# vi MY_GENERIC
```

Copy `XENHVM` into `MYKERNEL` file, then edit `MYKERNEL` file:

```
# cp XENHVM MYKERNEL
# vi MYKERNEL
```

Make the following changes in the file (remove the red lines and add the green lines without "+"):

```
-include        GENERIC
+include        MY_GENERIC

-ident      XENHVM
+ident      MYKERNEL

...
```

```
+# PF
+device pf
+device pflog
+device pfsync
```

Task 4. Build world and Custom Kernel

Type the code from Listing 2. When running *mergemaster* for the first time, please select "d" (delete "temporary file") for the following files:

```
/etc/group
/etc/master.passwd
```

Note

A "temporary file" is a file without your configuration changes, the one which comes with FreeBSD source.

If you are prompted for other files and these files contain some configuration changes you've done and you want to keep these changes, then you should also choose "d". If you want to replace your amended files with the default version from FreeBSD source, then select "i".

When you are running *mergemaster* for the second time (after `make installworld`), do not delete `ec2_*` files from `/etc/rc.d/` when asked. Same logic as above applies to this step. Be ready to answer many questions. It will take a while but you will be rewarded with the new custom kernel with all security patches applied.

Step 7. Setup OpenVPN

OpenVPN is a robust and highly flexible tunneling application that uses all of the encryption, authentication, and certification features of the OpenSSL library to securely tunnel IP networks over a single TCP/UDP port.

We will use OpenVPN on our FreeBSD NAT instance to make only one port open to public (UDP 1194). Once connected and authenticated, legitimate OpenVPN users will get access to all the servers behind the NAT instance, subject to the NAT instance's firewall rules.

Task 1. Install OpenVPN

Install OpenVPN from ports:

```
# cd /usr/ports/security/openvpn
# make config-recursive
# make install clean
```

Leave the default options when asked.

Listing 2. *Rebuilding "world" and custom kernel*

```
# cd /usr/src
# make cleanworld && make cleandir
# make buildworld
# make buildkernel KERNCONF=MYKERNEL
# make installkernel KERNCONF=MYKERNEL
# reboot
# adjkerntz -i
# mergemaster -p -U
# cd /usr/src
# make installworld
# make delete-old
# mergemaster -U
# reboot
# cd /usr/src
# make delete-old-libs
```

Listing 3. *Script to copy the sample configuration and the scripts to generate the certificates and keys*

```
# mkdir /usr/local/etc/openvpn
# cp /usr/local/share/doc/openvpn/sample-config-files/server.conf /usr/local/etc/openvpn
# cp -a /usr/local/share/doc/openvpn/easy-rsa /usr/local/etc/openvpn
# chmod 0755 /usr/local/etc/openvpn/easy-rsa/2.0/*
```

Listing 4. *Script for generating certificate for a user*

```
# cp /usr/local/etc/openvpn/keys/ca.crt /usr/local/etc/
                    openvpn/easy-rsa/2.0/keys/
# cp /usr/local/etc/openvpn/keys/ca.key /usr/local/etc/
                    openvpn/easy-rsa/2.0/keys/
# ./build-key user_name.your.domain
```

Listing 5. *OpenVPN server configuration file*

```
...
# MS Fix for Apps like Remote Desktop
fragment 1400
mssfix

# Which local IP address should OpenVPN
# listen on? (optional)
;local a.b.c.d
local 10.0.1.10
...
ca /usr/local/etc/openvpn/keys/ca.crt
cert /usr/local/etc/openvpn/keys/openvpn.your.domain.crt
key /usr/local/etc/openvpn/keys/openvpn.your.domain.key
                    # This file should be kept secret
...
dh /usr/local/etc/openvpn/keys/dh2048.pem
...
# Push routes to the client to allow it

# the server.  Remember that these
# private subnets will also need
# to know to route the OpenVPN client
# address pool (10.8.0.0/255.255.255.0)
# back to the OpenVPN server.
push "route 10.0.1.0 255.255.255.0"
push "route 10.0.2.0 255.255.255.0"
...
# Certain Windows-specific network settings
# can be pushed to clients, such as DNS
# or WINS server addresses.  CAVEAT:
# http://openvpn.net/faq.html#dhcpcaveats
# The addresses below refer to the public
# DNS servers provided by opendns.com.
push "dhcp-option DNS 8.8.4.4"
;push "dhcp-option DNS 208.67.220.220"
...
# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
# You can uncomment this out on
# non-Windows systems.
user nobody
group nobody
...
# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status /var/log/openvpn-status.log
...
# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
# Use log or log-append to override this default.
# "log" will truncate the log file on OpenVPN startup,
# while "log-append" will append to it.  Use one
# or the other (but not both).
;log         openvpn.log
log-append  /var/log/openvpn.log
…
```

Listing 6. *Configuration file for your Windows OpenVPN client*

```
client
dev tun
proto udp
remote openvpn.your.domain 1194
resolv-retry infinite
nobind

fragment 1400


persist-key
persist-tun

ca "d:\\path\\to\\directory\\ca.crt"
cert "d:\\path\\to\\directory\\user_name.your.domain.
                    crt"
key "d:\\path\\to\\directory\\user_name.your.domain.key"

comp-lzo
verb 3
```

Task 2.
Generate Certificates and Keys for OpenVPN Server
Now let's create a configuration directory, and then copy the sample configuration and the scripts we will need to generate the certificates and keys for our server and clients: Listing 3.

You can edit `/usr/local/etc/openvpn/easy-rsa/2.0/vars` file if you want to change the default RSA key size etc.

```
# /bin/sh
# cd /usr/local/etc/openvpn/easy-rsa/2.0 && . ./vars
# ./clean-all
```

Then create the CA:

```
# ./build-ca
```

When asked for the "Common Name" please specify the *Fully Qualified Domain Name* (FQDN) of your NAT instance (for example, `openvpn.your.domain`).

Generate the server key file:

```
# ./build-key-server openvpn.your.domain
```

When asked for the "Common Name" please again specify the *Fully Qualified Domain Name* (FQDN) of your NAT instance. Leave the Challenge Password empty.

Create the Diffie-Hellman parameters:

```
# ./build-dh
```

This can take a while...

Copy the files you've generated to the `/usr/local/etc/openvpn/keys` directory:

```
# mkdir /usr/local/etc/openvpn/keys
# cp /usr/local/etc/openvpn/easy-rsa/2.0/keys/*
/usr/local/etc/openvpn/keys
```

Then clean up everything from the `/usr/local/etc/openvpn/easy-rsa/2.0/keys` directory:

```
# ./clean-all
```

Task 3. Generate Certificates and Keys for OpenVPN Users
Generate certificate for a user: Listing 4. When asked for the Name and email address, please specify user's name and email address.

Each user certificate you create, needs to have a unique Common Name (CN). I usually use `user_name.your.domain` format for the Common Name to ensure the uniqueness. Leave the Challenge Password empty. Execute the above command for each user which will be connecting to your OpenVPN server. After that you should copy the user certificates to the `/usr/local/etc/openvpn/keys` directory:

```
# cp /usr/local/etc/openvpn/easy-rsa/2.0/keys/*
/usr/local/etc/openvpn/keys
# ./clean-all
```

The files you need to provide to each user are:

- `ca.crt`
- `user _ name.your.domain.crt`
- `user _ name.your.domain.key`

You can transfer the files by using WinSCP in the SCP mode. The key file (`user _ name.your.domain.key`) is accessible only by root. So you will need to change its permissions to make the transfer. But do not forget to revert it back once you are done. The file must be readable only by root.

Task 4. Configure OpenVPN server
Now let's configure OpenVPN server. Open `/usr/local/etc/openvpn/server.conf` and make the following changes: Listing 5. Enable the OpenVPN server during boot-time:

```
# echo 'openvpn_enable="YES"' >> /etc/rc.conf
# echo 'openvpn_configfile="/usr/local/etc/openvpn/server.
                    conf"' >> /etc/rc.conf
# echo 'openvpn_if="tun"' >> /etc/rc.conf
```

Start manually the OpenVPN server:

```
# /usr/local/etc/rc.d/openvpn start
```

Task 5. Configure OpenVPN Clients
Please find the configuration file for your Windows OpenVPN client in the Listing 6.

Note
The OpenVPN client can be downloaded from *http://openvpn.se/download.html*.

Step 8. Launch a Microsoft Windows Instance in the Private Subnet

Let's launch a Microsoft Windows instance in the private subnet to test that our NAT instance works properly:

- Select the “EC2” item from the “Services” menu in the AWS Management Console, and then click on the “Launch Instance” button.
- Choose the “Classic” wizard.
- Select “Microsoft Windows Server 2008 R2 Base 64-bit”.
- On the next wizard step choose the instance type, click on “VPC” and select your private subnet (10.0.2.0/24) from the list. Click “Continue”.
- Select “1” in the “Number of Network Interfaces” drop-down list, and setup 10.0.2.20 as the IP address (subnet 10.0.2.0/24).
- You can assign secondary IP addresses if needed. Then click “Continue” button twice.
- Name your instance and then click “Continue”.
- Create a new key pair or choose the existing one, and then click “Continue”.
- Create a new security group and open port 3389 for RDP (TCP). See Figure 13.
- Click “Launch”.

Step 9. Test Remote Desktop Connection to the Microsoft Windows Instance

Now let’s retrieve Windows password:

- Select the “EC2” item from the “Services” menu in the AWS Management Console.

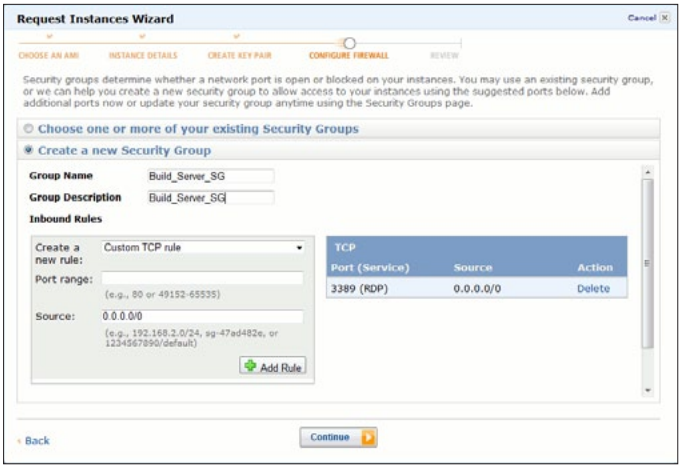


Figure 13. *Create a New Security Group for Windows Instance*

ICMP		
Port (Service)	Source	Action
Echo Reply	0.0.0.0/0	Delete
TCP		
Port (Service)	Source	Action
22 (SSH)	0.0.0.0/0	Delete
80 (HTTP)	10.0.2.20/32	Delete
443 (HTTPS)	10.0.2.20/32	Delete
UDP		
Port (Service)	Source	Action
1194	0.0.0.0/0	Delete

Figure 14. *Updated Security Group for NAT Instance*

- Click on the “Instances” on the “Navigation” pane.
- Right-click on the Windows instance and then select the “Get Windows Password” from the popup menu.
- Select the private key and then click on “Decrypt Password”.

Now you can connect to 10.0.2.20 using Remote Desktop Connection.

Please note that right now only RDC to the Microsoft Windows instance will work. Nothing else. If you try to browse the Internet or click on Windows Update, the Microsoft Windows instance will not be able to access the Internet. All its outgoing traffic goes through NAT instance but there are no such rules in the “NatSG” security group.

For example, to allow HTTP/HTTPS traffic from your Microsoft Windows instance, you should add the following rules to the NAT instance’s security group: Figure 14.

Conclusion

I hope that this article will motivate you to use FreeBSD instances in Amazon Cloud. The only downside of this is so called “Microsoft Windows tax”. You probably have noticed that all FreeBSD AMIs for non-cluster instances (at least the ones that are not EOL yet) run as Microsoft Windows instances. As the result, you have to pay more for using them. (You can find out more details about this on the *http://www.daemonology.net/blog/2012-01-16-FreeBSD-now-on-all-EC2-instance-types.html* webpage). Let’s hope that in the nearest future Amazon will provide a mechanism for running in HVM mode without being labeled as a “Windows” instance.

And once again, many thanks to Colin Percival (a FreeBSD developer, a member of the FreeBSD Core team, and the FreeBSD Security Officer) for his work to get FreeBSD running in Amazon Cloud.

VAND777

Andrey aka vand777 has been passionate about technology and computers since his early teens. He works for a financial institution in the UK. He likes to play with FreeBSD in his spare time running and managing approximately 5 FreeBSD operating system instances running name servers, mail servers, web servers, application servers, database servers, firewalls, etc. for some personal domains on the internet.

solutions

Open Source Systems
Design - Administration - Consulting

- design and administration of server farms
- load balancing and high availability solutions
- ZFS file servers and storage appliances
- FreeBSD, OpenIndiana and Linux

Our open source projects:

VX ConnectBot: SSH and telnet client for Android
mfsBSD: memory-resident FreeBSD installations
zfs-stats: ZFS statistics tools

Contact information:

VX Solutions s. r. o.
Mag. Martin Matuška
E-Mail: office@vx.sk
Web: http://www.vx.sk

PostgreSQL Indexes (Part 2)

This article continues the previous one, presenting the readers with a few index examples and how the access costs are computed by the query planner. All the examples shown here have been tested on a PostgreSQL 9.1 cluster running on a FreeBSD 8.2-RELEASE machine; all the example source code is available in a GitHub repository.

What you will learn...

- which indexes can be defined on existing data
- how to analyze the effectiveness of an index

What you should know...

- basic shell commands
- basic PostgreSQL concepts
- server-side programming with PostgreSQL

In this section a few example queries and indexes will be used to explain the basis of database indexes and how PostgreSQL performs the cost computation. It is assumed that the *articles* table has just been created and populated as described in the previous article in this series and that no custom indexes have been created.

First consider a single-column-filtered query: get all the articles of average difficulty (i.e., *difficulty = 'AVG'*) and see how PostgreSQL handles it (Listing 1).

The executor wants to perform a sequential scan (Seq Scan) that will cost 51667 and will extract 663933 tuples out of the original relation. The first step to understanding the generated plan is knowing how the system computed the number of output tuples: the system stores statistical data in the catalogues and make it available through the special view `pg_stats` (see Listing 2). Such statistical data is fundamental for the executor to choose the execution plan and the access method. In the case of the difficulty column the statistical data reports that there are only 3 distinct values in the whole table and that such values are 'MIN', 'MAX', and 'AVG', as expected (see the function in Listing 1).

The `most_common_vals and most_common_freqs` are arrays whose values are related, and therefore the 'MAX' value appears with a frequency of 0.3359, the 'MIN' value with a frequency of 0.332133 and the 'AVG' value with a frequency of 0.331967. This is the information required to know how many tuples the system is going to expect out of a query that filters on the difficulty = 'AVG' clause:

```
output_tuples    = relation_total_tuples * frequency
       = 2000000 * 0.331967
       = 663934
```

Listing 1. *A query when no indexes have been defined*

```
bsdmagdb=# EXPLAIN SELECT title, pages, listings, difficulty FROM articles WHERE difficulty = 'AVG';
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on articles  (cost=0.00..51667.00 rows=663933 width=38)
   Filter: (difficulty = 'AVG'::bpchar)
```

Now it is possible to compute the cost of the sequential scan: each data page of the target relation will be read with a weight of `seq_page_cost` and the system has to evaluate each tuple to see if it matches the selection condition, and therefore:

```
seq_scan_total_cost = relation_total_pages * seq_page_cost
          + relation_total_tuples * (cpu_tuple_cost + cpu_
                    operator_cost)
          = 26667 * 1 + 2000000 * (0.01+0.0025)
          = 51667
```

That is exactly the cost reported by the query planner.

In order to see how things change using indexes, build an index on the difficulty column as follows:

```
bsdmagdb=# CREATE INDEX idx_difficulty ON articles(difficulty);
```

Similarly to regular data tables, since PostgreSQL manages indexes as relations, it is possible to see the amount of space occupied by an index:

```
bsdmagdb=# SELECT relname, reltuples, relpages
FROM pg_class WHERE relname = 'idx_difficulty' AND relkind = 'i';
     relname     | reltuples | relpages
-----------------+-----------+----------
  idx_difficulty |     2e+06 |     4392
```

The index has the same tuple count of the data relation, and this is because the index is full (that is covers each data tuple). However, since the index stores a lot less information with regard to the data table the number of pages is less than that of the real data.

As shown in Listing 3 the planner now decides to exploit the newly created index and accesses the data table using such index. The first operation performed by the planner is a "Bitmap Index Scan", that is a fancy way to say "walk the index but don't get out the tuples yet, just keep the pointers"; in fact the output width for such node is 0 (that is no tuples at all). In this first pass therefore the planner walks the index to see which tuples match the difficulty = 'AVG' clause and builds a bitmap of pointers to such tuples. For efficiency reasons, if the number of matching tuples increases, then the bitmap does not keep tuple pointers but page data pointers, that is this pass is "lossy" and on the next step the planner knows that the condition has to be checked again (hence the "Recheck Cond" on the second node).

The second node reads the built bitmap and extracts the tuples out of the data table (the width now is not zero): it is interesting to note that the rows number of the two nodes are exactly the same (as expected). It is possible to compute the costs manually following similar steps as in the previous case:

Listing 2. *Listing 2. Statistical data used by the optimizer*

```
bsdmagdb=# \x
bsdmagdb=# SELECT tablename, attname, n_distinct, most_common_vals, most_common_freqs
FROM pg_stats WHERE tablename = 'articles' AND attname = 'difficulty';
tablename         | articles
attname           | difficulty
n_distinct        | 3
most_common_vals  | {MAX,MIN,AVG}
most_common_freqs | {0.3359,0.332133,0.331967}
```

Listing 3. *The query plan using an index on the "difficulty" column*

```
bsdmagdb=# EXPLAIN SELECT title, pages, listings, difficulty FROM articles WHERE difficulty = 'AVG';
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Bitmap Heap Scan on articles  (cost=10977.91..45944.07 rows=663933 width=38)
   Recheck Cond: (difficulty = 'AVG'::bpchar)
   ->  Bitmap Index Scan on idx_difficulty  (cost=0.00..10811.92 rows=663933 width=0)
         Index Cond: (difficulty = 'AVG'::bpchar)
```

```
output_tuples    = relation_total_tuples * frequency
      = 2000000 * 0.331967
      = 663934
```

The bitmap index scan has to traverse the whole index, and therefore has to read each data page of the index (sequentially, that is with `seq_page_cost`) and to apply a cpu operator on each of the tuples in the index; therefore the results are:

```
bitmap_index_scan    = ( index_total_pages * random_page_cost
          + index_total_tuples * ( cpu_index_tuple_cost +
                    cpu_operator_cost ) )
          * index_selectivity
          = ( 4392 * 4 + 2000000 * ( 0.005 + 0.0025 ) ) *
                    0.331967
          = ( 17568 + 15000 ) * 0.331967
          = 10811
```

and then comes the "Bitmap Heap Scan" cost (see Box 2), that is rounded to the cost of sequentially retrieve pages (skipping not-bitmapped tuples) and applying the filter operator (recheck):

```
bitmap_heap_scan     = ( relation_data_pages * seq_page_cost )
          + ( relation_tuples * index_selectivity * ( cpu_
                    tuple_cost + cpu_operator_cost )
          = ( 26667 * 1 ) + ( 2000000 * 0.331967 * ( 0.01
                    + 0.0025 ) )
          = 26667 + 8299
          = 34966
```

Note that the above is the cost of the "Bitmap Heap Scan" node which, as shown in Listing 3, has a final cost of around 45944 and an initial cost of around 10977 and therefore a single run cost of 45944 – 10977 = 34967 (some rounding in the results is acceptable). It is interesting to note that the initial cost of the Heap Scan node is not exactly the same of the final cost of the Index Scan node but is slightly higher; this is due to the sorting of the obtained bitmap and to the re-arrangement of the in memory structures so that the next node can start running.

Why, having an index on *difficulty*, does the system not use it to immediately retrieve tuples out of the target relation but performs this "fancy" two step algorithm? The idea is the index is not selective enough to justify a pure index scan (with the cost of `random_page_cost`) and that is better to build a pre-ordered map of which data pages are required and to access them sequentially.

Supposing that the *difficulty* = *'AVG'* is the normal database 'workload' (i.e., a query that needs to be optimized

Listing 4. *Creation of a partial index to match the workload query*

```
bsdmagdb=# UPDATE pg_index SET indisvalid = false WHERE indexrelid = 'idx_difficulty'::regclass;
bsdmagdb=# CREATE INDEX idx_difficulty_avg
ON articles(difficulty)
WHERE difficulty = 'AVG';
bsdmagdb=# SELECT relname, reltuples, relpages
FROM pg_class
WHERE relname = 'idx_difficulty_avg' AND relkind = 'i';
      relname       | reltuples | relpages
--------------------+-----------+----------
 idx_difficulty_avg |    666666 |     1466
```

Listing 5. *Executing the workload query with the partial index*

```
bsdmagdb=# EXPLAIN SELECT title, pages, listings, difficulty FROM articles WHERE difficulty = 'AVG';
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 Bitmap Heap Scan on articles  (cost=11031.27..46046.61 rows=667867 width=38)

   ->  Bitmap Index Scan on idx_difficulty_avg  (cost=0.00..10864.30 rows=667867 width=0)
         Index Cond: (difficulty = 'AVG'::bpchar)
```

since it is executed very often), then it is possible to improve performance by building a partial index that covers only this exact condition. Listing 4 shows the creation of the partial index and the size of the results: it is interesting to note that the index has now 1466 pages (against the 4392 of the full one) and a lot less tuples.

Listing 5 shows the execution plan to access the same data (i.e., difficulty = 'AVG') with the partial index in place. As readers can see the query plan has not changed, that is again a two step plan is selected, with a slightly different cost with respect to the previous case. The cost can be computed in a similar manner to the previous case, without having the index selectivity in the first node computation (since the index is 100% selective); what changes with respect to the previous case is the cost of the first node (Bitmap Index Scan):

```
bitmap_index_scan    = ( index_total_pages * random_page_cost
          + index_total_tuples * ( cpu_index_tuple_cost +
                    cpu_operator_cost ) )
          = ( 1466 * 4 + 666666 * ( 0.005 + 0.0025 ) )
          = ( 5864 + 5000 )
          = 10864
bitmap_heap_scan     = ( relation_data_pages * seq_page_cost )
          + ( relation_tuples * index_selectivity * ( cpu_
                    tuple_cost + cpu_operator_cost )
          = ( 26667 * 1 ) + ( 2000000 * 0.331967 * ( 0.01
                    + 0.0025 ) )
          = 26667 + 8299
          = 34966
```

Therefore, making a partial index on this particular query does not improve performance, and even if the resulting index is smaller, it still does not filter enough to be chosen as direct access method.

The Bitmap Index Scan is often better than a pure Index Scan, since the latter could imply reading the index randomly more than one time, while the Bitmap Index Scan sorts the bitmap in order to allow for sequential data retrieval. Therefore the Index Scan is used for very selective

Listing 6. *An Index Scan for a single row query retrieval*

```
bsdmagdb=# EXPLAIN SELECT title, abstract FROM articles WHERE pk = 1;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using articles_pkey on articles  (cost=0.00..8.44 rows=1 width=62)
   Index Cond: (pk = 1)
```

Listing 7. *A two-tuples query that uses a Bitmap Index Scan*

```
bsdmagdb=# EXPLAIN SELECT title, abstract FROM articles WHERE pk = 1 OR pk = 1000000;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Bitmap Heap Scan on articles  (cost=8.87..16.85 rows=2 width=62)
   Recheck Cond: ((pk = 1) OR (pk = 1000000))
   ->  BitmapOr  (cost=8.87..8.87 rows=2 width=0)
         ->  Bitmap Index Scan on articles_pkey  (cost=0.00..4.43 rows=1 width=0)
               Index Cond: (pk = 1)
         ->  Bitmap Index Scan on articles_pkey  (cost=0.00..4.43 rows=1 width=0)
               Index Cond: (pk = 1000000)
```

Listing 8. *A query similar to the previous one but with a different set of clauses*

```
bsdmagdb=# EXPLAIN SELECT title, pages, listings, difficulty
FROM articles WHERE difficulty <> 'MIN' AND difficulty <> 'MAX';
                               QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on articles  (cost=0.00..56667.00 rows=883552 width=38)
   Filter: ((difficulty <> 'MIN'::bpchar) AND (difficulty <> 'MAX'::bpchar))
```

queries (just a few rows). As an example consider a query that retrieves exactly one tuple: since the system builds a BTree for the primary key, the query of Listing 6 shows an Index Scan. The cost can be roughly computed as a `random_page_cost` for the index page and the data page, and therefore almost 8.

As an example, consider the query of Listing 7 that asks for two different rows via the table primary key: the optimizer chooses to perform a Bitmap Index Scan instead of an Index Scan. In such case the cost of the nodes can be computed in a very simple manner: having to access two different index entries on different index pages cost random_page_cost times two, and then comes the cost of retrieving the two data tuples (which could be in different pages), and therefore the overall cost is that of accessing four data pages at a random cost, and therefore 4 * 4 = 16. Then there is a little addition for the CPU costs and the need to do the *OR* between clauses, as shown in Listing 7.

It is interesting to note that the query planner can be confused by having the same query run with the wrong set of clauses: consider the plan for the query of Listing 8 that performs the same as the one of Listing 5 but without using the same indexes. The planner does not know that the {MIN, MAX} values are exclusive with regard to the {AVG} ones, and therefore performs a sequential scan with the already explained cost. This example emphasizes how often the DBA work is not only to find out the best access method to the data, but also to rewrite a "bad" query into one that the planner can better understand.

In order to explore more the PostgreSQL planner and index tools consider the creation of an index on the listings column and the query for all the tuples with the listing = 5 clause (with a frequency of 0.036633); this is a case very similar to the previous ones and as shown in Listing 9 the resulting index and value frequencies lead to a Bitmap Heap Scan.

```
bitmap_index_scan    = ( index_total_pages * random_page_cost
          + index_total_tuples * ( cpu_index_tuple_cost + cpu_
```

Listing 9. *Creation of an index on the listings column*

```
bsdmagdb# CREATE INDEX idx_listings
ON articles(listings);
bsdmagdb=# SELECT relname, reltuples, relpages
FROM pg_class
WHERE relname = 'idx_listings' AND relkind = 'i';
   relname    | reltuples | relpages
--------------+-----------+----------
 idx_listings |     2e+06 |     4392
bsdmagdb=# \x
bsdmagdb=# SELECT tablename, attname, n_distinct, most_common_vals, most_common_freqs
FROM pg_stats WHERE tablename = 'articles' AND attname = 'listings';
-[ RECORD 1 ]-----+-----------------------------------------------------------------------------------------------
tablename         | articles
attname           | listings
n_distinct        | 101

most_common_freqs | {0.3508,0.0380667,0.0371667,0.0371333,0.0368,0.0366667,0.0366333,0.0364,0.0361667,0.0335,0.0211
                  667}
bsdmagdb=# \x
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on articles  (cost=1212.24..28795.08 rows=73267 width=62)
   Recheck Cond: (listings = 5)
   ->  Bitmap Index Scan on idx_listings  (cost=0.00..1193.93 rows=73267 width=0)
         Index Cond: (listings = 5)
```

```
                operator_cost ) ) * index_selectivity
        = ( 4392 * 4 + 2000000 * ( 0.005 + 0.0025 ) ) *
                0.036633
        = ( 17568 + 15000 ) * 0.036633
        = 1193
bitmap_heap_scan    = ( relation_data_pages * seq_page_cost )
        + ( relation_tuples * index_selectivity * ( cpu_
                tuple_cost + cpu_operator_cost )
        = ( 26667 * 1 ) + ( 2000000 * 0.036633 * ( 0.01
                + 0.0025 ) )
        = 26667 + 916
        = 27583
total_cost      = 27583 + 1193 = 28776
```

How does the plan change if in a single query there are both clauses listings = 5 and difficulty = 'AVG'? The planner decides to use a direct index access, as shown in Listing 10. Why does the planner not use a Bitmap Heap Scan on one of available indexes? If the planner does, the index with the minimal cost would be the `idx_listings`, which would have a total cost of near 28776; then the cost for the filtering condition difficulty = 'AVG' has to be applied, and this results in a CPU operation applied on each output tuple of the index above, therefore increasing the cost of the whole access as follows:

```
bitmap_heap_scan    = 28776 + ( total_data_tuples * index_
                    selectivity * cpu_operator_cost )
        = 28776 + ( 2000000 *  0.036633 * 0.01 )
        = 29508
```

which is greater than the cost of a pure Index Scan (28188). The latter can be approximated to the worst case that is:

```
index_scan      = random_page_cost * total_index_pages
        + ( total_index_tuples * cpu_tuple_index_cost )
        * ( total_data_tuples * index_selectivity * (
                cpu_tuple_cost )
            + cpu_operator_cost * number_of_clauses )
        = 17568 + 10000 + 1098
        = 28666
```

It is possible to create a compound index that filters on both the clauses, as shown in Listing 11. Supposing a Bitmap Heap Scan, the cost of using such index for the query of Listing 10 would result as follows (note that the selectivity is total and that the index can be read sequentially):

```
bitmap_index_scan   = ( index_total_pages * seq_page_cost
        + index_total_tuples * ( cpu_index_tuple_cost +
```

Listing 10. *A query that has both the clauses*

```
bsdmagdb=# EXPLAIN SELECT title, pages, listings, difficulty
FROM articles WHERE difficulty = 'AVG' and listings = 5;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Index Scan using idx_difficulty_avg on articles  (cost=0.00..28188.14 rows=24561 width=38)
   Index Cond: (difficulty = 'AVG'::bpchar)
   Filter: (listings = 5)
```

Listing 11. *Creation of a compound index*

```
bsdmagdb=# CREATE INDEX idx_difficulty_avg_listings_5 ON articles(difficulty, listings)
WHERE listings = 5 AND difficulty = 'AVG';
bsdmagdb=# SELECT relname, reltuples, relpages
FROM pg_class

WHERE relname = 'idx_difficulty_avg_listings_5' AND relkind = 'i';
            relname             | reltuples | relpages
--------------------------------+-----------+----------
 idx_difficulty_avg_listings_5 |     66664 |      185
```

```
                    2 * cpu_operator_cost ) )
          = ( 185 * 1 + 66664 * ( 0.005 + 0.0025 ) )
          = ( 185 + 333 )
          = 518
bitmap_heap_scan     = ( relation_data_pages * seq_page_cost )
          + ( index_tuples * ( cpu_tuple_cost + 2 * cpu_
                    operator_cost )
          = ( 26667 * 1 ) + ( 66664  ( 0.01 + 2 * 0.0025 ) )
          = 26667 + 1000
          = 27667
total_cost       = 27667 + 518 = 28185
```

Therefore the total cost is nearly the same as the one obtained using another of previous indexes, and therefore it is not worth creating this index for this data set. This is confirmed by Listing 12, that shows how the planner decides to use the normal index instead of the compound one and how forcing the usage of the compound index does not provide a great improvement in performance.

Plans with a single relation can be easy enough to read and understand, but as more tables are involved in a set of joins the plan can become very complex. In order to ex-

Listing 12. *The cost of the access with the idx_listings and idx_difficulty_avg_listings_5 indexes*

```
bsdmagdb=# EXPLAIN SELECT title, pages, listings, difficulty
FROM articles WHERE difficulty = 'AVG' and listings = 5;                                    QUERY PLAN
-------------------------------------------------------------------------------------------
                    -
 Index Scan using idx_difficulty_avg on articles  (cost=0.00..28188.14 rows=24561 width=38)
   Index Cond: (difficulty = 'AVG'::bpchar)
   Filter: (listings = 5)
bsdmagdb=# set enable_indexscan TO false;
bsdmagdb=# EXPLAIN SELECT title, pages, listings, difficulty
FROM articles WHERE difficulty = 'AVG' and listings = 5;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
                    -
 Bitmap Heap Scan on articles  (cost=528.01..28100.18 rows=24561 width=38)
   Recheck Cond: ((difficulty = 'AVG'::bpchar) AND (listings = 5))
   ->  Bitmap Index Scan on idx_difficulty_avg_listings_5  (cost=0.00..521.87 rows=24561 width=0)
         Index Cond: ((difficulty = 'AVG'::bpchar) AND (listings = 5))
```

Listing 13. *A join query plan*

```
bsdmagdb=# EXPLAIN SELECT a1.title, a1.listings, a1.difficulty
FROM articles a1 JOIN articles a2 ON a1.pk = a2.pk
WHERE a1.difficulty = 'AVG'
AND   a2.listings = 5;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Hash Join  (cost=29961.92..72477.02 rows=24562 width=34)
   Hash Cond: (a1.pk = a2.pk)
   ->  Index Scan using idx_difficulty_avg on articles a1  (cost=0.00..26511.97 rows=670467 width=38)
         Index Cond: (difficulty = 'AVG'::bpchar)
   ->  Hash  (cost=28795.08..28795.08 rows=73267 width=4)
         ->  Bitmap Heap Scan on articles a2  (cost=1212.24..28795.08 rows=73267 width=4)
               Recheck Cond: (listings = 5)
               ->  Bitmap Index Scan on idx_listings  (cost=0.00..1193.93 rows=73267 width=0)
                     Index Cond: (listings = 5)
```

plain some join concepts consider a query that joins the *articles* table against itself in order to get all the articles with a *difficulty = 'AVG'* and a *listings = 5* as shown in Listing 13. The plan starts from the Bitmap Heap Scan (or better from the Bitmap Index Scan subnode) on the *idx_listings* index for the relation *a2* in order to get all the *listings = 5* tuples; the cost is the same computed in the previous examples for this kind of plan. The Bitmap Heap Scan produces a set of tuples made only by the attribute of the join clause, that is the *a2.pk* attribute (as confirmed by the length of 4 bytes, an integer value). The keys are then placed into an hash map in a similar way to what happens during a Bitmap Heap Scan (first phase). At the same time, on the relation *a1* an Index Scan is performed to extract all the *difficulty = 'AVG'* tuples; the cost is the same as computed in the previous examples. The output of this node is then compared with the whole hash map computed at the previous step in order to see if the tuple can be reported in output or not (see Figure 1). The cost of the whole plan depends on the *work_mem* size that should be large enough to keep in memory the whole hash map, otherwise multiple "batch" passes are required.

Statistics Target

The statistical data used by the query planner is computed depending on a configuration parameter called *statistics target* or *statistics* for short. Such parameter expresses how many data samples must be kept in the per-column statistics and therefore how the sample itself results accurate. Having a good statistical data is fundamental for the planner to take the right decision.

In order to better explain how the *statistics* work consider the *listings* column of the *article* table. As shown in Listing 9 the `most_common_vals` and `most_common_freqs` for the *listings* column includes all the ten values with their distribution, and this is the result of having a *statistics* greater or equal to the number of possible values. By default the *statistics* on each column is set to a value of 100 (since PostgreSQL 9), that means that no more than 100 samples of data will be kept. It is possible to change the value of the samples to be kept using the `ALTER TABLE ALTER COLUMN` command, as shown in Listing 14. For instance, setting the statics target to 5, as shown in Listing 14, produces the effect that the `most_common_values` and `most_common_freqs` of the *listings* column become a five-entries array therefore excluding some of the values that exist in the data set.

The side effect of this wrong statistical target is that a query plan could lead to incorrect results: the same query of Listing 9 (with the clause *difficulty = 5*) is still executed via the index *idx_listings* but the number of rows expected as output is very different from the real value (as shown in Listing 15): the planner believes that no more than 10240 rows match the clause, while the execution shows that 73309 rows do. As readers can imagine, having wrong statistical data will lead to a wrong query plan estimation that can lead to the wrong access path selection and therefore a non-optimal execution.

Of course it does not make sense to increase the number of statistical targets over the number of distinct values, and therefore for the *listings* column having more than 101 values does not affect the query planning but could affect the time of the `VACUUM ANALYZE` and other `ANALYZE` commands.

When and Where Placing an Index?

The most difficult job of a DBA is to understand when and where an index (of any kind) can improve query execu-

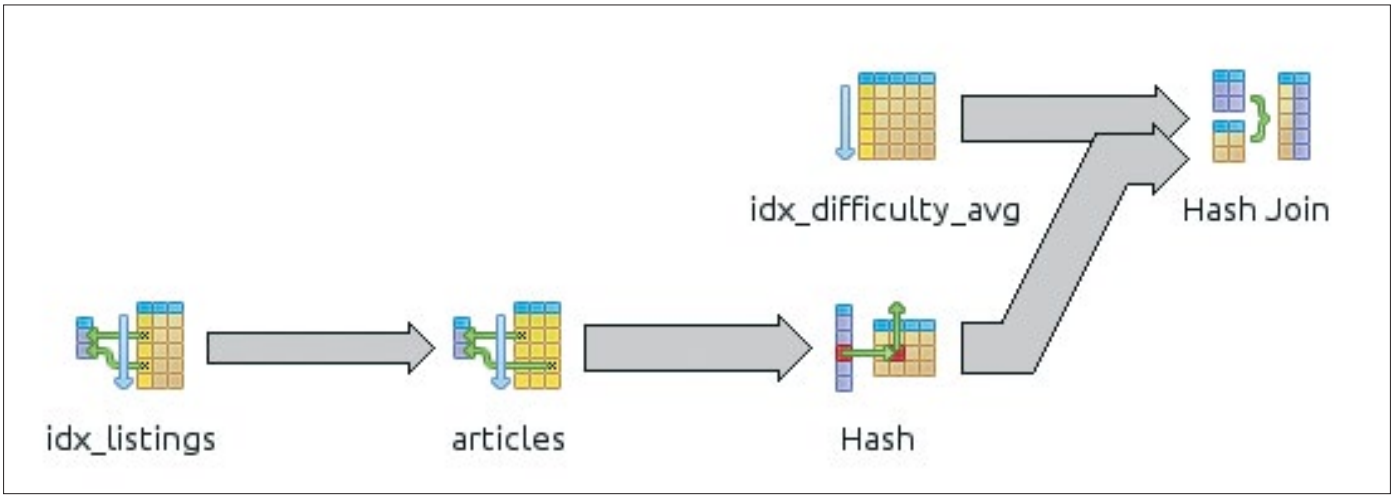


Figure 1. *A visual representation of the Hash Join (from PgAdmin III)*

tion. It is worth reminding that each index requires maintenance and can slow down `UPDATE`, `INSERT` and `DELETE` commands that need to modify such index. Therefore not used or not useful indexes should be removed from the database.

The first step in deciding when and where an index could be required is to identify slow queries, or better, queries that can slow down the whole application. In fact, it could be that a query that is executed quite fast is repeated so many times on different data (not cached) that the whole application performance results bad. The database administrator has to "observe" the database while it is working to get the most frequent queries or those that are too slow. There are different tools, like pgfouine (see Box 3), that can help the DBA analyze the database logs and report which queries are most often executed.

Out of the box PostgreSQL provides a configuration parameter, `log_min_duration_statement`, which can be used to log any query whose execution time requires more than the specified number of milliseconds. For instance setting the following:

```
log_min_duration_statement = 1000
```

Listing 14. *Decreasing the statistcs target*

```
bsdmagdb=# \x
bsdmagdb=# ALTER TABLE articles ALTER COLUMN listings SET STATISTICS  5;
bsdmagdb=# VACUUM FULL ANALYZE articles;
bsdmagdb=# SELECT tablename, attname, n_distinct, most_common_vals, most_common_freqs
FROM pg_stats WHERE tablename = 'articles' AND attname = 'listings';
-[ RECORD 1 ]-----+-------------------------------------------------
tablename         | articles
attname           | listings
n_distinct        | 101
most_common_vals  | {0,9,3,7,6}
most_common_freqs | {0.357267,0.0383667,0.0381,0.0376333,0.0371333}
```

Listing 15. *A query plan that has the wrong number of output rows due to wrong statistical target*

```
bsdmagdb=# EXPLAIN ANALYZE SELECT title, listings, difficulty
FROM articles WHERE listings = 5;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------
 Bitmap Heap Scan on articles  (cost=171.79..20035.29 rows=10240 width=34)
                               (actual time=124.177..1872.909 rows=73309 loops=1)
   Recheck Cond: (listings = 5)
   ->  Bitmap Index Scan on idx_listings  (cost=0.00..169.23 rows=10240 width=0)
                                          (actual time=106.112..106.112 rows=73309 loops=1)
         Index Cond: (listings = 5)
```

Listing 16. *An example of report on index usage*

```
SELECT indexrelname, idx_scan, idx_tup_read, idx_tup_fetch
FROM pg_stat_all_indexes
WHERE relname = 'articles';

------------------------------+----------+--------------+---------------
 idx_difficulty_avg           |        9 |      5421591 |       5421591
 idx_listings                 |        9 |       440064 |             3
 idx_difficulty_avg_listings_5  |        0 |            0 |             0
```

Box 1. Query hints

PostgreSQL does not allow DBAs and users to tell the executor which path to choose in order to perform a particular query. This kind of executor suggestions are known as "query hints" and are usually available on commercial databases. The reason why PostgreSQL does not support them is that usually query hints are used in a very bad way: typically a DBA hints the executor to use an index because he thinks that such index will solve any performance problem, but it turns out that the optimizer is better at doing the path selection job. Nevertheless, while it is not possible to hint the executor to use a specific index, it is possible to force it to not use an access method. This behavior is obtained using the tunables `enable_xxx` that can be found in the postgresql.conf configuration file and that can be set also on the psql command line as follows:

```
bsdmagdb=# SET enable_seqscan = 'off';
bsdmagdb=# SET enable_seqscan = 'on';
```

The available access method tunables are the following:

```
enable_bitmapscan
enable_hashagg
enable_hashjoin
enable_indexscan
enable_material
enable_mergejoin
enable_nestloop
enable_seqscan
enable_sort
enable_tidscan
```

Box 2. How is a Bitmap Heap Scan cost computed?

Assuming the simplest case of a single index scan (i.e., no loops), the cost of a Bitmap Heap Scan is computed pretty much as follows:

- compute the tuplesFetched on the data table as $tuplesFetched = reltuples \times indexSelectivity$
- compute the maximum number of pages of the data table T (one or greater);
- compute the number of pagesFetched using the Mackert and Lohman formula, so that if the number of tuples becomes greater the number of relation pages dominates, otherwise the number of tuples dominates:
$$pagesFetched = \frac{(2 \times T \times tuplesFetched)}{(2 \times T + tuplesFetched)}$$
- round `pagesFetched` so that it is not greater than T;
- if `pagesFetched` is less than 2 (very small number of tuples, that is the number of pages are dominating) the data page access cost is `randomPageCost`, otherwise the data page access cost is computed with the following formula that has the lower limit at `seqPageCost`:
$$costPerPage = randomPageCost - (randomPageCost - seqPageCost) \times \left(\frac{pagesFetched}{T}\right)^{0.5}$$
- so that the final I/O cost is done by the number of pagesFetched multiplied by the costPerPage and the overall cost is done by the I/O cost and the CPU cost:

$$totalCost = costPerPage \times pagesFetched + (tuplesFetched \times (cpuTupleCost + cpuOperatorCost \cdot numOfWhereClauses))$$

For more info see the source code of the function `cost_bitmap_heap_scan` in the file `src/backend/optimizer/path/costsize.c`.

Box 3. PgFouine

PgFouine is a tool that helps administrators find problematic queries exploiting the database logs; in fact the project defines itself as a "log analyzer". The software can be installed from the database/pgfouine ports or as a package as follows:

```
# pkg_add -r pgfouine
```

It is important to instrument the PostgreSQL cluster to log data in a way that PgFouine can understand; at the time of writing PgFouine accepts logs from syslog, stderr and in CSV format. An easy, even if not recommended, setup of PostgreSQL for a quick use of PgFouine is as follows (file postgresql.conf):

```
log_destination = 'stderr'
logging_collector = on
silent_mode = on
log_directory = 'pg_log'
log_filename = 'postgresql.log'
log_min_duration_statement = 0
log_line_prefix = '%t [%p]: [%l-1] '
```

that will produce a file `$PGDATA/pg_log/postgresql.log` with all the executed statement. It is then possible to analyze the result using PgFouine as follows:

```
> /usr/local/bin/pgfouine -file $PGDATA/pg_log/postgresql.log
                        -format text -logtype stderr
```

which will produce a text output similar to the following

```
#####  Overall statistics  #####
Number of unique normalized queries: 2
Number of queries:     12
Total query duration:  5.8s

#####  Queries by type  #####
SELECT:    12    100.0%

#####  Most frequent queries (N)  #####
    1) 10 - 0.0s - SELECT COUNT(*) FROM magazine;
2 - 5.8s - SELECT COUNT(*) FROM articles;

#####  Slowest queries (N)  #####
    1) 2.88 - 2 - SELECT COUNT(*) FROM articles;
0.00 - 10 - SELECT COUNT(*) FROM magazine;
```

The report from PgFouine shows that, since logging has been activated, the system has executed 12 SELECT statements and reports which ones were the most frequent and which the slowest. This information is very helpful for DBAs in order to catch problematic queries and their frequency.

On The Web

- PostgreSQL official Web Site: *http://www.postgresql.org*
- ITPUG official Web Site: *http://www.itpug.org*
- PostrgeSQL Explain Documentation: *http://www.postgresql.org/docs/current/static/sql-explain.html*
- GitHub Repository containing the source code of the examples: *https://github.com/fluca1978/fluca-pg-utils*
- PgFouine: *http://pgfouine.projects.postgresql.org/*

while produce a log entry like the following:

```
LOG:  duration: 1418.252 ms  statement: SELECT title,
listings, difficulty FROM articles WHERE listings = 5 AND
                    difficulty = 'AVG';
```

Depending on how fast such a query should run, the administrator can inspect further to see if an index can be used to improve the query execution speed.

Having identified slow or problematic queries will let the administrator take decisions on which indexes to build. Here come into play all the other commands, like `EXPLAIN` and `ANALYZE` (to adjust statistical data). It is possible to disable one access method (see Box 1) in order to test if an index of a particular type can provide a real performance gain or to see why it is not used (i.e., the cost of using such index). Moreover, it is possible to inspect the `pg_stat_all_index` catalog to get information about which indexes are used and how from the running system. Listing 15 shows an example of index reporting through the `pg_stat_all_indexes` where it is possible to see that the partial index `idx_difficulty_avg_listings_5` has never been used since its creation and therefore could be removed without altering actual performances.

Note

Please note that values extracted from `pg_stat_all_indexes` could be different depending on when the statistical data is extracted and which queries have been executed against the example data table).

The `idx_scan` column reports the total Index Scan, the `idx_tup_fetch` is the number of tuples retrieved through the index, and `idx_tup_read` is the index tuples read by index usage. All the numbers are absolute, that is are counting since the index creation. For instance, as of Listing 16, the `idx_listings` has been used mainly for Bitmap Index Scans, since the `idx_tup_fecth` is almost zero and the `idx_tup_read` is a lot greater; in other words the index has been walked but no data tuple has been retrieved directly using index references. On the other hand `idx_difficulty_avg` has been used for Index Scans, since the number of index tuples (i.e., index walking) and the number of data tuples retrieved coincide.

Summary and Coming Next

This article completed the glance at indexing and cost computation. Main tools to understand which decisions the query planner took to execute a query has been shown. It is worth noting that finding the query optimization problem is very complex and hard to solve, and experience is invaluable in this scenario. In the next articles other PostgreSQL features will be shown.

LUCA FERRARI

Luca Ferrari lives in Italy with his wife and son. He is an Adjunct Professor at Nipissing University, Canada, a co-founder and the vice-president of the Italian PostgreSQL Users' Group (ITPUG). He simply loves the Open Source culture and refuses to log-in to non-Unix systems. He can be reached on line at http://fluca1978.blogspot.com.

In the next issue:

- NetBSD 6.0 Release
- ZFS Root on FreeBSD
- Panoramic Photography with BSD
- Rehosting Former Platforms in Modern Ones

Next issue is coming in December!

FreeBSD Enterprise

Search with Apache Solr (Part 3)

Continuing with our series on Apache Solr we will look at crawling the bsdmag.org website with nutch.

What you will learn…

- How to set up Apache Nutch to crawl websites and inject the content into a MySQL database ans search with Solr

What you should know…

- BSD administration skills, FreeBSD Apache Solr Part 1 and Part 2 article

One of the important facets of enterprise search is to be able to search internal (Intranet) and external websites. On a smaller scale, it is relatively trivial to assemble some code in PHP or Perl to pull web pages from a site, extract the links from the HTML and then "wash, rinse, repeat". The difficulty arises when we want to index, rank, and effectively manage these results on a large scale. Almost 10 years ago, Apache Nutch was developed as the key technology to crawl 100 million webpages, and has proved time and again that it is an efficient scalable solution. Nutch can be clustered, it is robots.txt friendly, and using modular plug-ins and schemas, can be tuned to bias certain results first. While Nutch integration and tuning is quite specialized, it is fairly trivial to configure Nutch to dump results of a crawl session into MySQL (or any other JDBC based database for that matter), and rank / review these queries in Solr.

For more information about Apache Nutch, please see the Nutch Wiki, which can be found at *http://wiki.apache.org/nutch*.

Requirements

We will continue to develop our Solr installation from the last article. Another core will be added, and test data imported from a MySQL database. While I will be using the vi editor, substitute the editor of your choice if preferred (e.g. emacs, nano etc.).

Step 1. Get the Files

```
$ su
# pkg_add -r apache-ant
```

If you see:

```
pkg_add: warning: package 'apache-ant-1.8.2_1' requires
'javavmwrapper-2.3.5', but 'javavmwrapper-2.4' is installed
pkg_add: can't open dependency file '/var/db/pkg/jdk-
                    1.6.0.3p4_27/+REQUIRED_BY'!
dependency registration is incomplete
```

This dependency error can be ignored since apache ant runs under javavmwrapper-2.4.

Install wget and bash if they are not already installed:

```
# pkg_add -r wget
# pkg_add -r bash
```

Download Nutch:

```
# mkdir /tmp/nutch
# cd /tmp/nutch
# wget http://mirror.rmg.io/apache/nutch/2.1/apache-
                    nutch-2.1-src.tar.gz
# tar -xvzf apache-nutch-2.1-src.tar.gz
```

Step 2. Configure MySQL

```
# /usr/local/etc/rc.d/mysql-server stop
# cd /var/db/mysql
```

Restart MySQL and check for UTF support (Figure 1).

```
# mv my.cnf my.cnf-000
# cp /usr/local/share/mysql/my-small.cnf ./my.cnf
# vi my.cnf
```

Add under [mysqld] section:

```
character-set-server = utf8
collation-server=utf8_unicode_ci
# /usr/local/etc/rc.d/mysql-server start
# mysql -uroot
mysql> \s
```

Quit from MySQL

```
mysql> \q
```

Create a nutch.sql command file with the following contents: Listing 1.

Create the database used for Nutch:

```
# mysql < nutch.sql
```

Step 3. Configure Solr

Stop Tomcat and create a new collection

```
# /usr/local/etc/rc.d/tomcat7 stop
# cd /home/solr
# cp -R collection2 collection3
```

Edit `solr.xml` to reflect the new collection by adding the following lines: Listing 2.

Flush the index data:

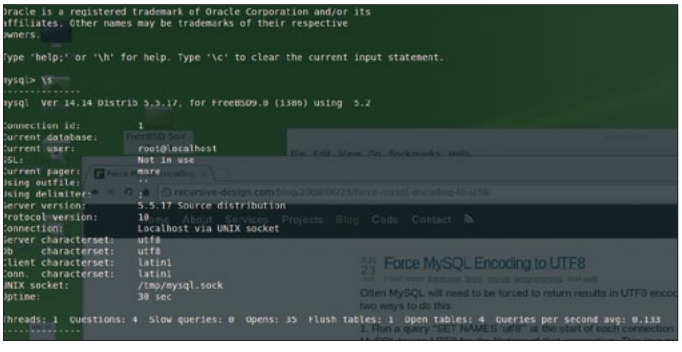


Figure 1. *MySQL with UTF8 support*

```
# rm collection3/data/index/*
# rm collection3/data/tlog /*
# cd collection3/conf
```

Modify the following files:

Listing 1. *SQL*

```
CREATE DATABASE nutch;
USE nutch;
CREATE TABLE 'webpage' (
'id' varchar(767) CHARACTER SET latin1 NOT NULL,
'headers' blob,
'text' mediumtext DEFAULT NULL,
'status' int(11) DEFAULT NULL,
'markers' blob,
'parseStatus' blob,
'modifiedTime' bigint(20) DEFAULT NULL,
'score' float DEFAULT NULL,
'typ' varchar(32) CHARACTER SET latin1 DEFAULT NULL,
'baseUrl' varchar(512) CHARACTER SET latin1 DEFAULT NULL,
'content' longtext DEFAULT NULL,
'title' varchar(2048) DEFAULT NULL,
'reprUrl' varchar(512) CHARACTER SET latin1 DEFAULT NULL,
'fetchInterval' int(11) DEFAULT NULL,
'prevFetchTime' bigint(20) DEFAULT NULL,
'inlinks' mediumblob,
'prevSignature' blob,
'outlinks' mediumblob,
'fetchTime' bigint(20) DEFAULT NULL,
'retriesSinceFetch' int(11) DEFAULT NULL,
'protocolStatus' blob,
'signature' blob,
'metadata' blob,
PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

GRANT SELECT,INSERT,UPDATE ON nutch.* TO
                    'nutch'@'localhost' IDENTIFIED BY
                    'password';
```

Listing 2. *XML*

```
<core schema="schema.xml"
   instanceDir="/home/solr/collection3/"
   name="collection3"
   config="solrconfig.xml"
   dataDir="/home/solr/collection3/data"
/>
```

`data-config.xml`

This should contain the following: Listing 3.

schema.xml

Remove everything between the `<fields>` tags and delete all `<copyField>` configuration so that the schema.xml looks like: Listing 4.

solrconfig.xml

Modify the data import handler thus:

```
<str name="config">/usr/home/solr/collection3/conf/data-
                    config.xml</str>
```

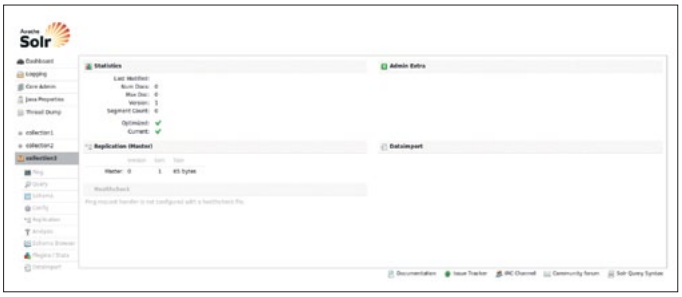


Figure 2. *Solr empty collection 3*

Figure 3. *Solr collection 3 schema*

Figure 4. *Solrconfig.xml for collection3*

Ensure that the `/etc/hosts` file has a valid host name assigned to local host otherwise nutch will fail:

```
127.0.0.1               localhost solr
```

Reboot:

```
# reboot
```

Check that the hosts file is correct:

```
$ ping -t3 solr
```

You should get a reply.

Check that you have a collection with no documents indexed by accessing the solr management interface with your browser at *http://youripaddress:8080/solr* (replace youripaddress with the IP address of your SOLR box).

The schema and data import handler should look like this: Figure 3.

Listing 3. *XML 2*

```
<dataConfig>

 <dataSource driver="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost/nutch"
             user="nutch"
             password="password"
    />

 <document name="webpage">

 <entity name="webpage"
       query="select ID,TITLE,STATUS,BASEURL,CONTENT,M
                    ETADATA from webpage">
          <field column="ID" name="id" />
         <field column="TITLE" name="title" />
         <field column="STATUS" name="status" />
         <field column="BASEURL" name="baseurl" />
         <field column="CONTENT" name="content" />
         <field column="METADATA" name="metadata" />
 </entity>

</document>
</dataConfig>
```

Step 4. Build and Configure Nutch

```
$ su
# cd /tmp/nutch/apache-nutch-2.1
```

Edit the `ivy/ivy.xml` file removing comments on lines 105 and 107 enabling MySQL support so line 105 is as follows:

```
<dependency org="mysql" name="mysql-connector-java"
                    rev="5.1.18" conf="*->default"/>
```

Edit the `conf/gora.properties` file and comment out the default Sqlstore properties, adding MtSQL as the data store so the file reflects: Listing 5.

Copy the nutch defaults across for our site crawl:

```
cp conf/nutch-default.xml conf/nutch-site.xml
```

Edit line 63 to reflect a sensible user agent as this cannot be empty:

```
<value>FreeBSD_Test</value>
```

Build nutch:

```
# ant runtime
```

This will take a few minutes as the jar files etc. are downloaded and nutch built.

```
# cp -R runtime/local /home/solr/nutch
# cd /home/solr/nutch
```

Edit the first line of `bin/nutch` to match the following:

```
#!/usr/local/bin/bash
```

Listing 4. *XML 3*

```
<fields>
   <field name="id" type="text_general" indexed="true" stored="true" />
   <field name="title" type="text_general" indexed="true" stored="true" />
   <field name="status" type="int" indexed="true" stored="true" />
   <field name="baseurl" type="text_general" indexed="true" stored="true"/>
   <field name="content" type="text_general" indexed="true" stored="true"/>
   <field name="metadata" type="text_general" indexed="true" stored="true"/>
   <field name="text" type="text_general" indexed="true" stored="true"      multiValued="true"/>
 </fields>

 <uniqueKey>id</uniqueKey>

 <copyField source="id" dest="text"/>
 <copyField source="title" dest="text"/>
 <copyField source="status" dest="text"/>
 <copyField source="baseurl" dest="text"/>
 <copyField source="content" dest="text"/>
 <copyField source="metadata" dest="text"/>
```

Listing 5. *XML 4*

```
#gora.sqlstore.jdbc.driver=org.hsqldb.jdbc.JDBCDriver
#gora.sqlstore.jdbc.url=jdbc:hsqldb:hsql://localhost/nutchtest
#gora.sqlstore.jdbc.user=sa
#gora.sqlstore.jdbc.password=

gora.sqlstore.jdbc.driver=com.mysql.jdbc.Driver
gora.sqlstore.jdbc.url=jdbc:mysql://localhost:3306/nutch?  \ createDatabaseIfNotExist=true
gora.sqlstore.jdbc.user=nutch
gora.sqlstore.jdbc.password=password
```

Add `JAVA_HOME` at the beginning of the file before `cygwin=false`

```
JAVA_HOME=/usr/local
```

Set up the nutch seed file (Replace *http://bsdmag.org/* if desired):

```
# mkdir -p urls
# echo 'http://bsdmag.org/' > urls/seed.txt
```

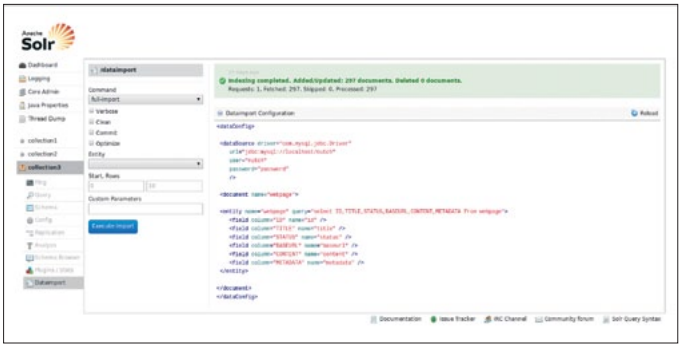


Figure 5. *Successful data import for collection3*

Figure 6. *Search query for Apache Solr*

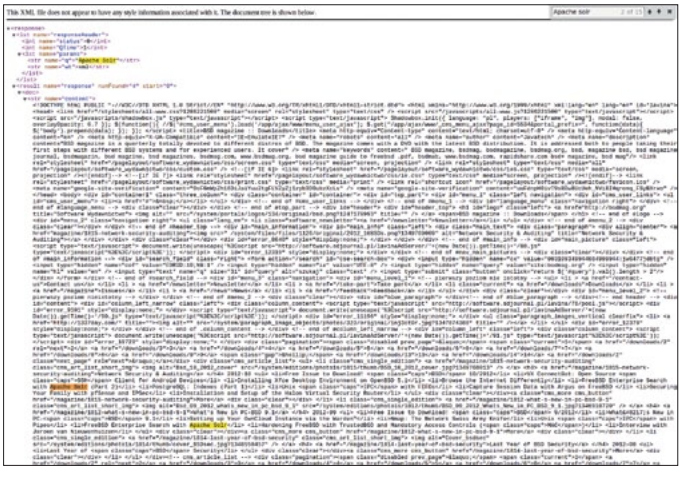


Figure 7. *Content from http://bsdmag.org*

Step 5. Crawl!

```
# bin/nutch crawl urls -depth 3 -topN 5 > & logs/crawl.log
```

Open another terminal:

```
$ cd /home/solr/nutch/logs
$ su
# tail -f crawl.log
```

You should see the site being crawled:

```
fetching http://bsdmag.org/magazine/1815-network-security-auditing
10/10 spinwaiting/active, 5 pages, 0 errors, 0.3 0.2
pages/s, 57 54 kb/s, 5 URLs in 1 queues
fetching http://bsdmag.org/news
10/10 spinwaiting/active, 6 pages, 0 errors, 0.2 0.2 pages/s,
61 78 kb/s, 4 URLs in 1 queues
* queue: http://bsdmag.org
  maxThreads    = 1
  inProgress    = 0
  crawlDelay    = 5000
  minCrawlDelay = 0
  nextFetchTime = 1348457886871
  now           = 1348457884591
  0. http://bsdmag.org/newsletter
  1. http://bsdmag.org/take-part
```

When complete (About 7 minutes as we are not crawling with any depth) run the following:

```
# mysql -unutch -ppassword
mysql> use nutch;
mysql> select count(*) from webpage where status = 2;
```

This should show the number of successfully fetched pages.

Conclusion

This is a very crude configuration, but should be enough to get you started. More attention needs to be paid to the schema files (in particular to clear out redundant cruft and to optimize).

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

Register Today!

November 3rd & 4th

The event is being held at YAHOO! in Sunnyvale, CA

Registration is available at www.MeetBSD.com/registration

MeetBSD California 2012 promises to be an experience unlike any other.

MeetBSD California is not your average conference - it's a meeting of the minds from all over the BSD community. MeetBSD California 2012 will feature community - driven break - out sessions, discussion groups, and 5 -10 minute "lightning talks," as well as longer talks from seasoned BSD experts.

@MeetBSDCA

www.facebook.com/MeetBSDCalifornia